普通高等教育“十一五”国家级规划教材
新编 21 世纪高等职业教育电子信息类规划教材・电气自动化技术专业

MCS-51 单片机原理及接口技术
（第 2 版）

汪德彪　　主　编

钟秉翔
刘显荣　　副主编

胡文金　　主　审

電子工業出版社

Publishing House of Electronics Industry

北京・BEIJING

内 容 简 介

本书以 MCS-51 系列单片机为背景介绍单片机原理及其接口技术，主要内容包括：单片机组成结构；单片机指令系统和汇编语言程序设计；单片机中断技术和定时器/计数器应用；单片机功能扩展技术；C51 编程语言及其应用；键盘、显示、A/D、D/A、开关量等功能性接口技术；单片机通信技术；单片机应用系统设计方法、步骤、以及工程应用举例等。

本书编写突出应用性和实践性，将单片机应用中的最新技术写入教材。在接口技术部分，将 SPI、I^2C 接口技术作了详细介绍，并有实际例子帮助读者加深理解 SPI、I^2C 技术。在通信部分，把重点放在 PC 机与单片机之间点对点和点对面的通信技术方面，具有较高的实用价值。将 C51 写入教材是对单片机程序设计语言的最好补充和延伸，C51 部分的许多例子都有与之对应的汇编语言例子，便于读者对比学习。教材中的例子都取材于工程应用，突出例程的实用性和完整性，这不仅仅是为了学习知识而举例，而更重要的是为了掌握单片机的基本应用特性，从应用出发，又回到实际应用中去。

本书深入浅出，淡化理论，突出工程应用，适合作为高职高专院校自动化及其相关专业的教材，也可作为工程技术人员的参考用书。

图书在版编目(CIP)数据

MCS-51 单片机原理及接口技术/汪德彪主编. —2 版. —北京：电子工业出版社，2009.6
新编 21 世纪高等职业教育电子信息类规划教材. 电气自动化技术专业
ISBN 978-7-121-08989-3

Ⅰ. M… Ⅱ. 汪… Ⅲ. ① 单片微型计算机—基础理论—高等学校：技术学校—教材② 单片微型计算机—接口—高等学校：技术学校—教材 Ⅳ. TP368.1

中国版本图书馆 CIP 数据核字（2009）第 088972 号

策　　划：陈晓明
责任编辑：陈晓明　　特约编辑：王宝祥
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1 092　1/16　印张：18　字数：461 千字
印　　次：2009 年 6 月第 1 次印刷
印　　数：4 000 册　　定价：27.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。
服务热线：（010）88258888。

前　言

随着我国现代化进程的不断推进，高级技术应用型人才的社会需求不断增加，高等职业教育承担着培养技术型人才的重任越来越受到国家的高度重视。高等职业教育的人才培养目标是培养生产、建设、服务、管理一线的应用型人才，突出实用性与针对性。单片机课程作为电气电子类专业的主要技术课程受到各院校高度重视，纷纷进行课程改革和实验室建设，广大学生利用单片机进行电子设计制作并参加电子设计大赛，极大地增强了学生工程实践能力，这正体现了单片机技术的应用特性。对于学生来讲，掌握单片机知识与技能后，可以直接凭此参与就业。鉴于此，本书在内容选取上注重应用性，淡化理论，把在工程实际中广泛应用的知识、技术讲透，并辅助以工程实际例子来加强应用性，举例力求其完整性；在内容组织上注意高职院校学生的特点，深入浅出，循序渐进。

本书共分为 9 章，第 1 章为 MCS-51 单片机的基本知识和组成结构；第 2 章为 MCS-51 单片机指令系统及其程序设计；第 3、4、5、6、7 章为 MCS-51 单片机的中断技术、定时器/计数器、功能扩展、接口技术和串行通信技术等；第 8 章为 C51 及其应用编程，以简明扼要的方式介绍 C51 的基本规则，举例大多是前面章节汇编语言程序的翻版，对照性强，便于掌握。目前 Keil C/Franklin C 已被工程技术人员广泛采用，使得单片机应用系统的开发效率大大提高，通过这一章的学习，基本能达到使用 C51 编程的目的；第 9 章讲述单片机应用系统设计开发的步骤、方法，以及单片机应用系统的抗干扰技术，并有三个完整的应用实例，其中电动小车自平衡系统为 2007 年全国大学生电子设计竞赛题目，具有综合性，而简易电脑时钟具有趣味性，全部程序用 C51 编写。

本书由汪德彪担任主编，负责第 1 章编写并对全书进行统稿，副主编钟秉翔编写第 2 章和第 8 章，副主编刘显荣编写第 4 章和第 5 章，叶文编写第 6 章，张义辉编写第 3 章，李家庆和李芳编写第 9 章，刘解生编写第 7 章。要特别感谢四川工程职业技术学院郭杰和天津滨海职业技术学院王玉松，他们在本书第一版编写中做出了巨大贡献。还要特别感谢在本书第一版使用中提出宝贵意见的胡文金、彭宇兴等同志，同时，胡文金教授担任本书主审，提出了很多非常好的修改意见。本书在编写过程中得到重庆科技学院电子信息工程学院领导和教务处领导的关心和支持，在本书统稿过程中得到唐德东同志的许多宝贵意见，在此表示衷心感谢。

由于编者水平有限，书中必定存在疏漏和错误，不足之处在所难免，恳请读者批评指正。

编　　者

2009 年 3 月于重庆

目　录

第 1 章　MCS-51 系列单片机的组成

本章主要内容

单片机概述。MCS-51 单片机的基本结构。MCS-51 单片机的存储器结构与 I/O 口。MCS-51 单片机的工作方式。

1.1　单片机概述

计算机系统的发展已明显地朝巨型化、单片化、网络化三个方向发展。巨型化发展的目的在于不断提高计算机的运算速度和处理能力以解决复杂系统计算和高速数据处理，比如系统仿真和模拟、实时运算和处理。单片化是把计算机系统尽可能集成在一块半导体芯片上，其目的在于计算机微型化和提高系统的可靠性，这种单片计算机简称单片机。单片机的内部硬件结构和指令系统主要是针对自动控制应用而设计的，所以单片机又称微控制器 MCU（Micro Controller Unit），用它可以很容易地将计算机嵌入到各种仪器和现场控制设备中，因此单片机又叫做嵌入式微控制器（Embedded MCU）。单片机自 20 世纪 70 年代问世以来，以其鲜明的特点得到迅猛发展，已广泛应用于家用电器、智能玩具、智能仪器仪表、工业控制、航空航天等领域，经过 30 多年的发展，性能不断提高、品种不断丰富，已经形成自动控制的一支中坚力量。据统计，我国的单片机年产量已达 1～3 亿片，且每年以大约 16%的速度增长，但相对于世界市场我国的占有率还不到 1%。这说明单片机应用在我国有着广阔的前景。对于从事自动控制的技术人员来讲，掌握单片机原理及其应用已经成为其不可或缺的学习任务。

1.1.1　单片机的发展历史

随着大规模集成电路的出现，诞生了微处理器，也才有了单片机的产生与发展，单片机的发展大致经历了以下几个阶段：

第一阶段（1974 年～1976 年）为单片机的初级阶段。由于受工艺和集成度的限制，这一阶段单片机的主要特点是功能和结构简单，片内只包含 8 位的 CPU、64B 的随机存储器和 2 个并行 I/O 接口。受制造水平和工艺限制，采用双片结构，还需外接一片内含 ROM、定时器/计数器和并行 I/O 接口的集成电路才能构成微型计算机，严格意义上讲，这还不是单片机。

第二阶段（1976 年～1980 年）为低性能单片机阶段。在这一阶段里，单片机的结构和性能逐步得到提高，但总体性能依然比较低，单片机技术逐渐走向成熟。1976 年，Intel 公司研制生产的 MCS-48 系列单片机是第一代通用单片机，这一系列单片机内含 8 位 CPU、64B RAM、定时器/计数器、27 条 I/O 引线，这一系列单片机中有些产品，如 P8748H 内含 1KB EPROM，使得开发应用方便不少。正是由于这一通用单片机的推出，开辟了单片机市场，促进了单片机技术的迅猛发展。

第三阶段（1980 年～1983 年）为高性能单片机阶段。在这一阶段，超大规模集成电路

使得单片机技术更加成熟，虽然 CPU 仍然是 8 位，但其运算速度已经提高到 12MHz，芯片内的 ROM 和 RAM 容量大大增加，I/O 线的数量也达到 32 条，单片机的性能成倍提高。这一时期的典型产品有 MCS-51 系列单片机和 MC68C05 系列单片机等。

第四阶段（1983 年以后）为功能完善阶段。同普通微型计算机一样，单片机从 8 位发展到 16 位乃至 32 位，如 MCS-96 系列单片机就是高性能 16 位单片机。但由于 8 位单片机已经能满足大多数用户的需求，以其价廉物美优势，牢牢占据市场的主导地位。近年来，单片机向着高性能、多品种的方向发展。一是 CPU 功能不断增强，增加乘法、除法部件，采用流水线结构，运算速度快、精度高。二是内部资源增多，如增加 A/D、PWM、HSIO、FlashROM、RAM、甚至 DMA 功能。三是使用多功能引脚，功能增加而引脚数量相对减少，提高了应用的灵活性。四是低功耗和低工作电压，采用 CHMOS 工艺，增加了空闲等待方式和停机方式，工作电压降低，干电池足以使其长时间工作，便于携带和野外作业使用。

1.1.2 目前主流单片机

单片机经过 30 多年的发展，已经形成一个规格齐全、品种繁多的大家族，用户有非常大的选择余地。下面为读者简单介绍目前市面上常见的主流单片机。

1. MCS-51 单片机及其兼容产品

Intel 公司于 1980 年推出 8 位的高性能 8051 单片机，在工业控制领域引起不小轰动，并迅速确立了其不可动摇的地位。之后不久，Intel 公司彻底开放了 8051 单片机的技术，引来世界上很多半导体厂商加入了开发和改造 8051 单片机的行列中，这其中贡献最大的有 Philips 公司，它着力发展了单片机的控制功能和外围单元；Atmel 公司，它在单片机内部植入了 FlashROM，使得单片机应用变得更灵活，在我国拥有大量的用户；ADI 公司，它推出的 ADuC8xx 系列单片机在单片机向 SOC 发展的模/数混合集成电路发展过程中扮演了很重要的角色；Cygnal 公司采用一种全新的流水线设计思路，使单片机的运算速度得到极大提高，使单片机在向 SOC 发展的过程中迈出了一大步。不管这些厂商对 8051 单片机进行了如何改进，但都保留了 8051 原有的指令系统和内部的基本结构，因此所有这些单片机的指令是相互兼容的，开发工具也大致相同，业界把这类单片机统称为 MCS-51 系列单片机。

经过众多国际大公司的共同开发和支持，MCS-51 系列单片机以其品种最齐全、开发手段最丰富、技术资料和程序资源最完备等特点早早确立了其作为工业控制标准单片机的地位，所以本书将重点向读者介绍 MCS-51 系列单片机的知识。

表 1.1 列出了部分当前比较常见的 MCS-51 系列单片机，供读者选型参考。

表 1.1 常见的 MCS-51 系列单片机

型　号	工作电压（V）	速度/频率	Flash ROM	OTP ROM	RAM	UART	定时器	I/O 口	描　述
Intel 公司部分产品									
8031	5	12MHz			128	1	2	32	片内无 ROM
8051	5	12MHz		4K	128	1	2	32	
8751	5	12MHz			128	1	2	32	4K 片内 EPROM
8052	5	12MHz		8K	256	1	3	32	

续表

型　号	工作电压（V）	速度/频率	Flash ROM	OTP ROM	RAM	UART	定时器	I/O 口	描　述
Atmel 公司部分产品									
AT89C2051	3～5	24MHz	2K		128	1	2	15	20 脚 PDIP、SOIC 封装，带比较器
AT89C51	5	33 MHz	4K		128	1	2	32	Intel8051 的换代产品
AT89C52	5	33MHz	8K		256	1	3	32	40 脚 PDIP、PLCC44 封装
AT89S53	5	24MHz	12K		256	1	3	32	带 ISP 和 SPI 接口
CYGNAL 公司部分产品									
C8051F000	2.7～3.6	20MIPS	32K		256	1	4	32	带 I2C、SPI 接口、12×8 位 ADC、12 位 DAC、电压基准和温度传感器
C8051F001	2.7～3.6	20MIPS	32K		256	1	4	16	带 I2C、SPI 接口、12×8 位 ADC、12 位 DAC、电压基准和温度传感器
C8051F020	2.7～3.6	25MIPS	32K		4352	2	5	64	带 I2C、SPI 接口、12×4 位 ADC、12 位 DAC、电压基准和温度传感器
C8051F302	2.7～3.6	25MIPS	8K		256	1	3	8	I2C 接口、8×8 位 ADC、温度传感器
Philips 公司部分产品									
P87C52	4.5～5.5	33MHz		8K	256	1	3	32	
P87LPC762	4.5～5.5	16MHz		16K	512	1	2	18	带 I2C 接口
P87LPC767	4.5～5.5	20MHz		4K	128	1	2	18	带 I2C 接口，8×4 位 ADC
P89C660	4.5～5.5	20MHz	16K		512	1	3	32	带 I2C 接口，ISP
P89C662	4.5～5.5	20MHz	16K		1024	1	3	32	带 I2C 接口，ISP
P87C51MC2BA	3/5	24MHz		96K	3072	1	3	32	16M 外部数据访问空间
ADI 公司部分产品									
ADuC812BS	3/5	16MHz	8K		256	1	3	32	8×12 位 ADC，2×12 位 DAC 640byteEEPROM
ADuC816BS	3/5	16MHz	8K		256	1	3	32	2×16 位 ADC，1×12 位 DAC 640byteEEPROM，可编程增益输入
ADuC814BRU	3/5	16MHz	8K		256	1	3	11	6×12 位 ADC，2×12 位 DAC 640byteEEPROM，28 脚封装低功耗

2. 其他单片机介绍

当很多公司在改造 MCS-51 系列单片机的同时，世界上一些有影响力的大公司也在开发

自己的单片机，比如 Motorola、TI、Microchip、Atmel、OKI、ST、Epson 等。这些单片机的指令系统和内部结构都和 MCS-51 系列单片机不同，功能也各有千秋。

（1）Motorola 的单片机。Motorola 是世界上最早开发单片机的著名厂商，是目前全球最大的 8 位单片机生产商。现在已经拥有 8 位、16 位和 32 位约十几个系列的单片机，这其中 8 位机主要有 68HC05、68HC08 和 68HC11 等 3 个系列；16 位机主要有 HCS12、68HC12、DSP56800 和 68HC16 等 4 个系列；32 位机主要有 Coldfire 的 MC683xx、MCORE、MPC500 和 MCF5xxx 等系列。

Motorola 单片机的功能一般都很强，进入我国的时间也很早，在单片机应用领域有很高威望，但由于其开发工具价格较高，影响了普及率。

（2）Microchip 公司的 PIC 单片机。Microchip 公司是当今世界第二大 8 位单片机生产商，Microchip 单片机在我国也有比较多的用户，近几年随着 Microchip 不断推出颇具特色的各型单片机，Microchip 已越来越受到业界的广泛关注。目前，市面上比较常见的单片机主要有：PIC12C5xx、PIC16C5xx、PIC16Fxxx 等系列。PIC12C5xx、PIC16C5xx 这两个系列的单片机是 PIC 单片机中的低端产品，其中 PIC16C5xx 系列是最早在市场上得到发展的系列，因其价格较低，且有较完善的开发手段，因此在国内应用最为广泛；而 PIC12C5xx 是世界第一个 8 脚低价位单片机，可用于一些对单片机体积要求较高的简单智能控制领域，前景十分广阔。PIC12C6xx/PIC16Cxxx 系列是 PIC 中档产品，是 Microchip 近年来重点发展的系列产品，品种最为丰富，其性能比低档产品有所提高，增加了中断功能、指令周期可达到 200ns、带 A/D、内部 E^2PROM 数据存储器、双时钟工作、比较输出、捕捉输入、PWM 输出、I^2C 和 SPI 接口、异步串行通信（USART）、模拟电压比较器及 LCD 驱动等等，其封装从 8 脚到 68 脚，可用于高、中、低档的电子产品设计中，价格适中，广泛应用在各类电子产品中。PIC17Cxx 系列是 PIC8 位单片机中的高档产品，适合高级复杂系统开发的系列产品，其性能在中档位单片机的基础上增加了硬件乘法器，指令周期可达成 160ns，它是目前世界上 8 位单片机中性价比最高的机种之一，可用于高、中档产品的开发，如马达控制、音调合成。

（3）TI（Texas Instruments）公司的 MSP430 单片机。TI 公司是闻名全球的 DSP 制造商，但其前几年才推出的 MSP430 系列 16 位单片机同样在业界掀起不小的波澜。MSP430 系列单片机最突出的特点是低电压供电和超低功耗，非常适合应用于采用电池长时间工作的场合。电压范围为 1.8V～3.6V；在 1MHz 2.2V 下，活动模式功耗为 225μA、待机模式功耗为 0.8μA、掉电模式功耗为 0.1μA。在这个系列中有很多个型号，它们是由一些基本功能模块按照不同的应用目标组合而成。MPS430 系列单片机的 CPU 采用 16 位 RISC 精简指令系统，集成有 16 位寄存器和常数发生器，发挥了最高代码效率；它采用数字控制振荡器（DCO），使得从低功耗模式到唤醒模式的转换时间小于 6μs；内部集成了 A/D 转换器，方便工业应用；其中 MSP430x41x 系列设计有一个 16 位定时器、一个比较器、96 段 LCD 液晶驱动器和 48 个通用 I/O 口。

1.1.3 单片机的应用领域

单片机以其体积小、速度快、功耗低、使用灵活、价格低廉等特点，广泛应用于国民经济各个领域，对各行各业的技术改造和产品更新换代起着重要的推动作用。其主要应用领域可以概括为以下几个方面。

1. 工业控制

单片机作为一种工业控制计算机，主要用于简单工业控制系统或大型工业控制系统的前端，如过程控制（温度、压力、流量、液位控制）、数控机床、工业机器人等。

2. 智能仪器仪表

单片机一出现就成为改造传统电动仪器仪表最有力的工具，从而使仪器仪表智能化成为可能，大量功能完善的智能化仪器仪表也应运而生。如智能化数字存储示波器、智能化电工仪表、智能化自动化仪表等。

3. 数据采集与处理

单独扩充 A/D 功能，形成数据采集系统，对采集数据进行前期处理，为构成大型数据采集系统提供基础，如数据采集卡或数据采集器等。也可单独处理数据信息，形成智能数据处理设备，如数字传真机、智能化打印机、智能化绘图仪等。

4. 家用电器

家用电器一直是单片机应用的重要领域，竞争也十分激烈，低档、中档、高档产品层出不穷，如洗衣机、电冰箱、空调器、热水器、电饭煲、电视机、音响、电子玩具等，都广泛使用单片机进行控制。

航空航天也是单片机的重要应用领域，如飞行器控制、火箭制导、遥测遥控、无线通信等系统中都广泛应用单片机。

1.2 MCS-51 系列单片机的内部结构

单片机的应用要靠硬件和软件结合才能发挥其作用，我们在使用单片机时必须先弄清楚单片机的结构和内部的可用资源。

1.2.1 微处理器结构

单片机的内部结构如图 1.1 所示。由图可知，MCS-51 单片机片内主要由振荡电路、中央处理器（CPU）、内部总线、程序存储器、数据存储器、定时器/计数器、串行口、中断系统和 I/O 口等模块组成，各部分通过内部总线紧密地联系在一起。

（1）运算器。包括算术逻辑部件 ALU、布尔处理器、累加器 ACC、B 寄存器、两个暂存器和 BCD 码调整电路等组成。其作用主要包括：

① 加、减、乘、除算术运算。

② 增量（加 1）、减量（减 1）运算。

③ 十进制数调整。

④ 位置 1、清 0 和取反。

⑤ 与、或、异或等逻辑操作。

⑥ 数据传送操作。

（2）布尔处理器。布尔处理器是 CPU 中运算器的一个重要组成部分，它有相应的指令系统，硬件有自己的累加器（C）和自己的位寻址 RAM 以及 I/O 空间。

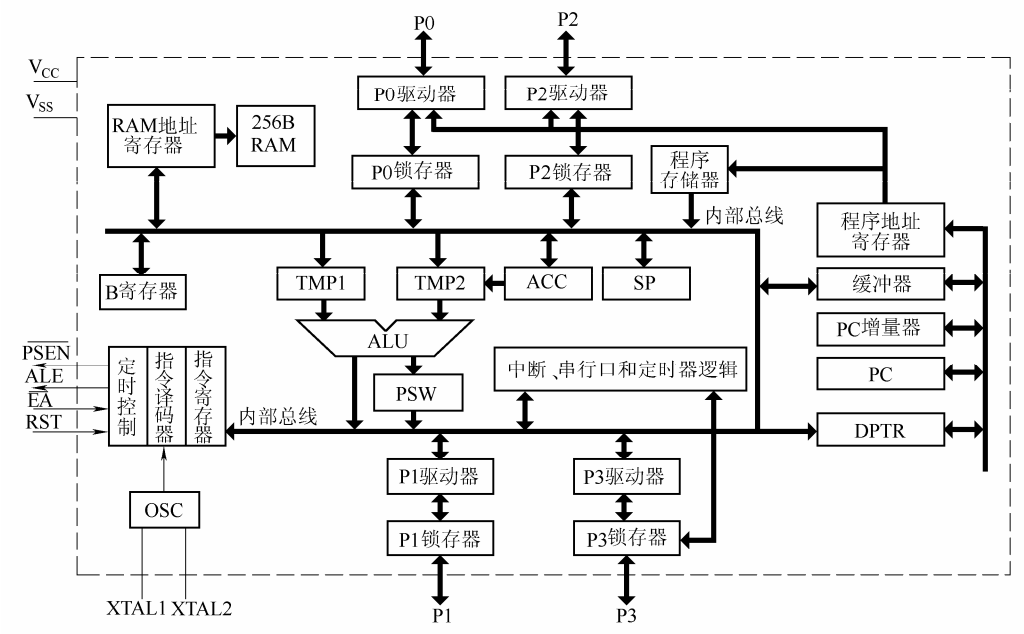


图 1.1　MCS-51 单片机的内部结构

（3）控制器。包括时钟电路、复位电路、定时控制逻辑、指令寄存器、指令译码器、程序指针 PC、数据指针 DPTR、堆栈指针 SP 和程序状态字寄存器 PSW 等。其作用主要是：

① 控制单片机内部各单元的协调工作。

② 协调单片机与外围芯片或设备的工作。

其中，程序指针 PC 的内容永远指向 CPU 正在执行指令的下一条指令在程序存储器中的单元位置，即改变 PC 的值就改变程序的执行方向。

程序状态字寄存器 PSW，它的内容反映 CPU 对数据处理的一些状态结果和对工作寄存器区的选择标志位。

Cy	AC	F0	RS1	RS0	OV	—	P

P：奇偶标志位。当累加器 ACC 中的处理结果数据中有奇数个“1”时为 1；否则为 0。

OV：溢出标志位。当 CPU 对数据处理结果发生溢出时，该位为 1；否则为 0。关于溢出概念将在后面的学习中具体讲解。

RS1 RS0：工作寄存器区选择位。

当（RS1RS0）=00B 时，第 0 工作寄存器区为当前区。

当（RS1RS0）=01B 时，第 1 工作寄存器区为当前区。

当（RS1RS0）=10B 时，第 2 工作寄存器区为当前区。

当（RS1RS0）=11B 时，第 3 工作寄存器区为当前区。

F0：用户标志位，通过指令可将其置 1 或清 0。

AC：辅助进位标志位。数据处理过程中低 4 位向高 4 位有进位或借位时，该位为 1；否则为 0。

Cy：进位标志位，当数据处理过程中最高位有进位或借位时，该位为 1；否则为 0。

1.2.2 振荡与时钟

单片机必须在时钟的驱动下才能工作。MCS-51 系列单片机内部有一个时钟振荡电路，只需外接振荡源，就能产生一定频率的时钟信号送到单片机内部的各个单元，决定单片机的工作速度。

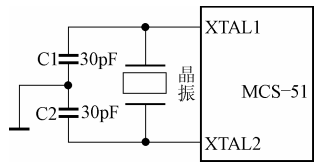


图 1.2 内部振荡时钟电路

图 1.2 就是内部钟工作方式的电路图，这是一种常用的方式。这种方式是外接振荡源，一般选择石英晶体振荡器。此电路在加电后延迟大约 10ms 振荡器起振，在 XTAL2 引脚产生幅度为 3V 左右的正弦波时钟信号，其振荡频率主要由石英晶振的频率确定。电路中两个电容 C_1、C_2 的作用有两个：一是帮助振荡器起振（C_1、C_2 值大，起振速度慢；C_1、C_2 值小，起振速度快）；二是对振荡器的频率起微调作用（C_1、C_2 值大，频率略有降低，C_1、C_2 值小，频率略有提高）。C_1、C_2 的典型值为 30pF。

MCS-51 单片机的工作时钟也可以从外部送入，这种情况比较少用，对于 HMOS 型和 CHMOS 型的时钟引入的方法是不一样的，请读者参阅相关书籍。

1.2.3 时钟周期、状态周期和机器周期

CPU 的工作就是不断地取指令和执行指令，以完成数据的传送、运算和输入/输出等操作。CPU 取出一条指令到该指令执行结束所需要的时间称为指令周期。不同的指令其指令周期不同，指令周期是以机器周期为单位来衡量时间的长短。

1. 时钟周期

单片机在工作时，由内部振荡器产生或由外部直接输入的送至内部控制逻辑单元的时钟信号的周期称为时钟周期。其大小是时钟信号频率的倒数，常用 f_{osc} 表示。如时钟频率为 12MHz，即 f_{osc}=12MHz，则时钟周期为 1/12μs。时钟周期又称为振荡周期或节拍，分为 P1 节拍和 P2 节拍，P1 节拍通常用来完成算术逻辑操作，P2 节拍通常用来完成内部寄存器之间的数据传送。

2. 状态周期

一个状态周期 *S* 由两个时钟周期构成。

3. 机器周期

一个机器周期由六个状态周期（S_1～S_6）或者说由十二个时钟周期构成。很显然，如时钟频率为 12MHz，即 f_{osc}=12MHz，则机器周期为 1μs。

时钟周期、状态周期和机器周期之间的关系如图 1.3 所示。

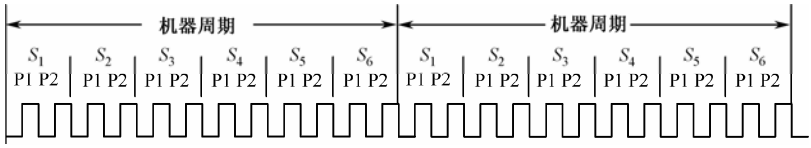


图 1.3 时钟周期、状态周期和机器周期之间的定时关系

1.2.4 复位及复位电路

1．复位状态

计算机在启动运行时都需要复位，复位使中央处理器 CPU 和系统中的其他器件都处于一种初始状态，并从这个初始状态开始工作。MCS-51 系列单片机有一个复位引脚 RST。

在 MCS-51 系列单片机的 RST 引脚上输入一个高电平信号，该高电平信号至少要维持两个机器周期（或者是 24 个时钟周期）以上的时间，单片机被复位。为了可靠地复位，复位时间一般都在 10ms 以上。当 RST 引脚变为低电平时，单片机退出复位，CPU 从初始状态开始工作。

MCS-51 系列单片机复位后内部各单元的初始状态，如表 1.2 所示。

表 1.2　复位后单片机内部各单元的初始状态

寄　存　器	初始状态值	寄　存　器	初始状态值
PC	0000H	TMOD	00H
ACC	00H	TCON	00H
B	00H	TH0	00H
PSW	00H	TL0	00H
SP	07H	TH1	00H
DPTR	0000H	TL1	00H
P1、P2、P3、P4	0FFH	SCON	00H
IP	×××00000B	PCON	0×××0000B
IE	0××00000B	SBUF	不定

复位使特殊功能寄存器 SFR 的内容归于复位值有着重要意义。程序计数器 PC=0000H，表示单片机执行程序的起始位置为 0000H 地址；P0～P3=0FFH，表示 P0～P3 口为准双向口，可以进行输入或输出操作；PSW=00H，表示选择 0 组工作寄存器，所有状态标志清 0；SP=07H，表示堆栈单元从 08H 开始等等。这些 SFR 的内容含义将在后面的学习中逐步分析，在此不再赘述。

2．复位电路

与其他计算机一样，MCS-51 单片机系统的复位方法有上电自动复位、手动复位以及“看门狗”复位等。

（1）上电自动复位电路。对于 MCS-51 单片机，在 RST 复位引脚端接一个电容至+5V 和一个电阻至地端，就能实现上电自动复位，如图 1.4 所示。在加电瞬间，电容通过电阻充电，就在 RST 引脚上出现一定时间的高电平信号，只要高电平信号时间足够长，就可以使 MCS-51 单片机有效地复位。RST 引脚在加电时应保持的高电平时间包括+5V 的上升时间和振荡器起振的时间，所以一般为了可靠地实现复位，$R1C$ 时间常数应取得大一些，当振荡频率为 12MHz 时，典型值为 C=10μF，R_1=8.2kΩ。

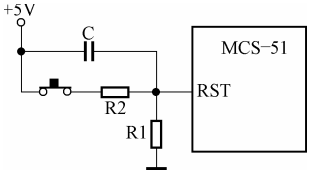


图 1.4　上电/手动复位电路

（2）手动复位电路。在系统运行过程中，有时可能对系统需要进行复位，为避免对硬件经常加电和断电造成的损害，我

们可以采用手动复位，如图 1.4 所示。这种方法是将一个开关串联一只电阻 R_2 后，再并联于上电自动复位的电容两端，在加电时自动复位，而在系统运行过程中需要复位时只要使开关闭合，在 RST 引脚上就会出现一定时间的高电平信号，从而使单片机实现复位。

（3）“看门狗”复位电路：在单片机应用系统中，单片机由于受到各种干扰而不能正常工作，在程序运行过程中出现死机是普遍存在的问题。为了尽快恢复单片机系统的功能，就要求将单片机复位而重新开始工作。目前单片机广泛应用于家用电器、汽车电控、工业自动化控制等系统中，单片机能否可靠上电复位和如何有效对程序运行进行监控，关系到单片机系统设计的成败。Motorola、Microchip、三菱、东芝等许多公司为提高单片机运行的可靠性，相继开发了内置有电压检测和看门狗定时器功能的单片机。但在我们实际使用过程中，发现这些内置的功能并不理想。尤其在电源电压波动比较大和电源开关比较频繁的场合，往往不能可靠上电复位甚至出现死机。因此，如何设计一个电路，保证单片机可靠上电复位，以及对程序运行进行监控，特别是对 MCS-51 系列内部没有看门狗定时器的单片机显得尤为重要。目前市面上已有专门的微机电源监控集成电路，如 MAXIM 公司的 MAX703～709/813L，它们具有系统复位、备份电源切换、“看门狗”定时输出、电源电压监测等多项功能。如图 1.5 所示，由 MAX813L 构成的 MCS-51 单片机的电源监测、按键复位和“看门狗”复位的电路。

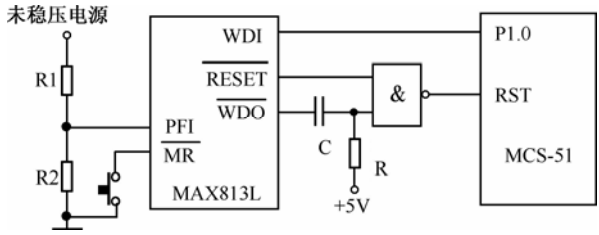


图 1.5　电压监测和按键、上电、“看门狗”复位电路

在电路中，MCS-51 单片机通过 P1.0 给 MAX813L 的 WDI 引脚加正脉冲，只要两次脉冲之间的时间间隔不大于 1.6s，则 $\overline{\text{WDO}}$ 引脚永远是高电平，单片机正常工作。如果单片机死机，则单片机不能在 1.6s 内清除“看门狗”定时，于是 MAX813L 的 $\overline{\text{WDO}}$ 输出低电平，经过与非门后使单片机复位。该电路还具有电源电压监测、按键复位、上电自动复位的功能。R_1 和 R_2 对未稳压的电源进行分压，当 PFI 上的电压下降到 1.25V 时，产生 RESET 信号，使单片机系统复位。同理，按键按下时也能产生复位信号。为了保证单片机可靠复位，可选取 R=10kΩ，C=10μF。

1.2.5　MCS-51 系列单片机的引脚特性

MCS-51 系列单片机采用 40 引脚双列直插式（DIP40）封装形式，如图 1.6 所示，有的采用 44 引脚 PLCC 和 TQFP 封装，各管脚的功能如下：

P0.0～P0.7：为 P0 口的 8 位引脚，不进行系统扩展时作为并行 I/O 口使用。进行系统扩展时往往作为地址/数据总线来使用。

P1.0～P1.7：为 P1 口的 8 位引脚，通常只作为 I/O 口使用。

P2.0～P2.7：为 P2 口的 8 位引脚，不进行系统扩展时作为并行 I/O 口使用。进行系统扩展时往往作为高 8 位地址总线来使用。

P3.0～P3.7：为 P3 口的 8 位引脚，除作为 I/O 口使用外，还有第二功能，详见表 1.3。

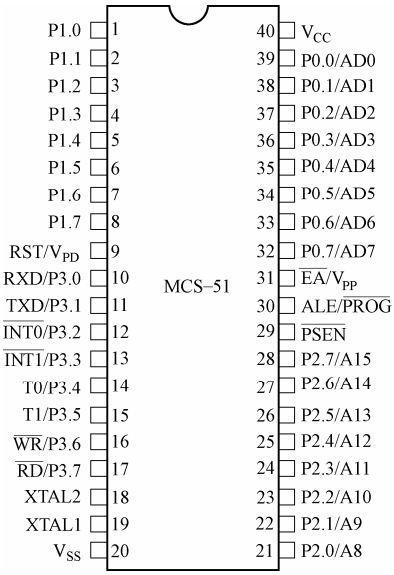


图 1.6　MCS-51 单片机引脚排列

RST/V_{PD}：为单片机的复位信号输入端/后备电源输入端。

$\overline{PSEN}$：为外部程序存储器读选通输出信号端。系统从片外程序存储器空间取指令执行指令时输出一个有效信号，此信号一般用于外部程序存储器读取指令的选通信号，以区别读取外部数据存储器。

XTAL2：为内部振荡电路反相器的输出端。

XTAL1：为内部振荡电路反相器的输入端。

ALE/$\overline{PRG}$：为地址锁存允许输出信号/编程脉冲信号输入端。此信号为外部的扩展电路低位字节地址提供一个允许锁存信号和对芯片内的 EPROM/EEPROM 空间固化程序时提供一个编程脉冲信号。

$\overline{EA}$/V_{PP}：为片内程序存储器选择输入信号端/编程电压输入端。当$\overline{EA}$=1 时，片内程序存储器有效；当$\overline{EA}$=0 时，片内程序存储器无效。对于内部有程序存储器的单片机，$\overline{EA}$=1；对于内部无程序存储器的单片机，必须是$\overline{EA}$=0。

V_{SS}：为芯片工作接地端。

V_{CC}：为芯片工作电源输入端（+5V）。

表 1.3　MCS-51 单片机 P3 口的第二功能

引　脚	名　称	第二功能说明
P3.0	RXD	串行数据输入
P3.1	TXD	串行数据输出
P3.2	$\overline{INT0}$	外部中断 0 请求输入
P3.3	$\overline{INT1}$	外部中断 1 请求输入
P3.4	T0	定时器/计数器 0 计数脉冲输入
P3.5	T1	定时器/计数器 1 计数脉冲输入
P3.6	$\overline{WR}$	外部数据存储器空间写信号输出
P3.7	$\overline{RD}$	外部数据存储器空间读信号输出

1.3　MCS-51 系列单片机的存储器组织

MCS-51 系列单片机有五个独立的存储器空间：

64KB 的程序存储器空间（0000～FFFFH）。

128 字节内部 RAM 空间（00H～FFH）。

128 字节内部特殊功能寄存器空间（80H～FFH）。

位寻址空间（00H～FFH）。

64KB 外部数据存储器空间（0000H～FFFFH）。

MCS-51 系列单片机的存储器结构如图 1.7 所示。特别值得注意的是：MCS-51 系列单片机各型号芯片在各个存储器空间的物理单元数据是不同的。

1.3.1 程序存储器

MCS-51 系列单片机对外是通过 P0、P2 口的口线形成地址总线，一共是 16 位，所以 MCS-51 单片机的最大程序存储器空间为 64KB，其地址指针为 16 位的程序计数器 PC。各个单片机的主要区别之一是片内是否有程序存储器，以及单元数的多少。

对于片内有 4KB 程序存储器空间的单片机（如 AT89C51），若 $\overline{EA}$=1 时，则程序计数器 PC 的值在 0000H～0FFFH 之间时，CPU 先从片内程序存储器空间取指令执行，当 PC 的值大于 0FFFH 时才访问外部的程序存储器空间。若 $\overline{EA}$=0 时，则片内的程序存储器空间被忽略，CPU 总是从片外程序存储器空间取指令执行。

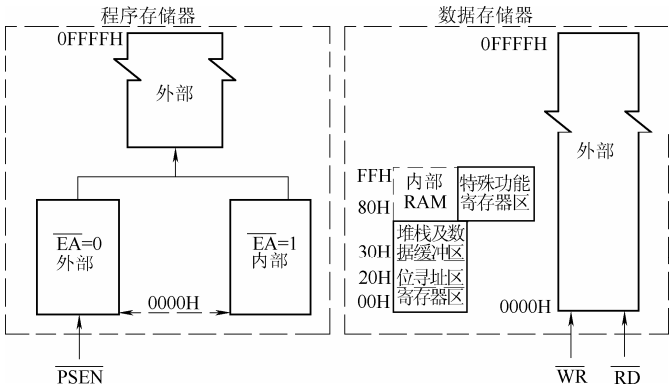


图 1.7　MCS-51 单片机的存储器结构

对于片内有 8KB 程序存储器空间的单片机（如 AT89C52），若 $\overline{EA}$=1 时，则程序计数器 PC 的值在 0000H～1FFFH 之间时，CPU 先从片内程序存储器空间取指令执行，当 PC 的值大于 1FFFH 时才访问外部的程序存储器空间。若 $\overline{EA}$=0 时，则片内的程序存储器空间被忽略，CPU 总是从片外程序存储器空间取指令执行。

对于片内无程序存储器的单片机（如 8031、8032 等）在构成系统时必须在外部扩展程序存储器，其引脚 $\overline{EA}$ 必须接地。

MCS-51 系列单片机中提供了一个外部程序存储器选通输出信号引脚 $\overline{PSEN}$，仅当 CPU 访问片外程序存储器时，$\overline{PSEN}$ 才输出一个有效的低电平信号。系统中片外扩展的程序存储器一般为 EPROM、EEPROM、以及 FlashROM。

MCS-51 系列单片机程序存储器空间的 7 个特殊单元的功能如表 1.4 所示。

表 1.4　MCS-51 单片机特殊程序存储器单元的功能

地　址	功　能
0000H	程序执行的起始地址
0003H	外部中断 0 的中断服务程序入口地址
000BH	定时器/计数器 0 的中断服务程序入口地址
0013H	外部中断 1 的中断服务程序入口地址
001BH	定时器/计数器 1 的中断服务程序入口地址
0023H	串行口的中断服务程序入口地址
002BH	定时器/计数器 2 的中断服务程序入口地址（仅 52 子系列）

从表 1.4 中可以看出，0000H 到 0003H 只有 3 个字节的间隔，仅能存放 1 条转移指令，因此一般在程序存储器空间开始，存放 1 条无条件转移指令，让程序转移到主程序。从 0003H 到 002BH 单元，是 6 个中断源的服务程序入口地址，它们之间的间隔也只有 8 个字节，也只能存放几条指令，因此往往在中断服务程序开始安排 1 条转移指令，转移到另一个合适的地方进行中断处理。当然，在这些特殊存储单元没有被使用时，也可以作为普通的程序存储单元。

1.3.2 片内数据存储器

MCS-51 系列单片机内部数据单元是弥足珍贵的资源，这些存储单元功能多样，应用灵活，其容量为 256B，地址范围为 00H～FFH，实际提供给用户使用的 RAM 容量随型号不同而有所差异。51 子系列只有 128B，52 子系列才有 256B。

MCS-51 单片机的内部数据存储器低端的 128B 根据功能和用途可以划分为三个区：工作寄存器区、位寻址区、堆栈及数据缓冲区。

1．工作寄存器区

片内数据存储器的地址为 00H～1FH 区域为四个工作寄存器区，每个区有 8 个单元，每个单元分配了一个工作寄存器编号 R0～R7。工作寄存器与 RAM 单元地址之间的对应关系如表 1.5 所示。

CPU 当前使用的工作寄存器区是由程序状态字寄存器 PSW.3（RS0）和 PSW.4（RS1）来决定，对应关系如表 1.6 所示。CPU 可以通过修改 RS1 和 RS0 两位的值就能选择一个工作寄存器区作为当前区。这一点为 CPU 进行现场保护、现场恢复和数据处理提供了更多的单元，提高了 CPU 的工作效率。如果在实际系统中不需要 4 个工作寄存器区，多余的工作寄存器区单元可以作为一般的数据缓冲区使用。工作寄存器区只能按单元形式访问，而且一般是以工作寄存器标识符的形式对单元访问。

表 1.5　工作寄存器与 RAM 单元地址对照表

单元地址	符　号	区名称
00H	R0	第 0 工作寄存器区
01H	R1	
……	……	
07H	R7	
08H	R0	第 1 工作寄存器区
09H	R1	
……	……	
0FH	R7	
10H	R0	第 2 工作寄存器区
11H	R1	
……	……	
17H	R7	
18H	R0	第 3 工作寄存器区
19H	R1	
……	……	
1FH	R7	

表 1.6　工作寄存器区选择对应表

PSW.4（RS1）	PSW.3（RS0）	当前工作寄存器区（R0～R7）
0	0	0 区（00H～07H）
0	1	1 区（08H～0FH）
1	0	2 区（10H～17H）
1	1	3 区（18H～1FH）

2．位寻址区

片内 RAM 单元的 20H～2FH 为位寻址区域，这 16 个单元的每一位都可以单独地进行位操作，且每一位都分配了一个位访问地址（00H～7FH），位寻址区各位的位地址分配如表 1.7 所示。在应用中每一位可以作为一个软件触发器，用于存放各种程序标志、位控制变量等。如果系统中不使用位寻址功能，该区也可以作为一般的数据缓冲区使用。位寻址区既可以按位访问，也可以按字节访问。

我们看到：在内部 RAM 区中，低 128 单元地址范围是 00H～7FH，而位寻址区的各位的位地址范围是 00H～7FH，二者是重叠的，在应用中如何区分位地址和单元地址呢？这一点取决于指令的类型。

表 1.7　RAM 位寻址区地址映像表

单元地址	位　地　址							
	D7	76	D5	D4	D3	D2	D1	D0
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	77H	76H	75H	74H	73H	72H	71H	70H
2DH	6FH	6EH	6DH	6CH	6BH	6AH	68H	68H
2CH	67H	66H	65H	64H	63H	62H	61H	60H
2BH	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	57H	56H	55H	54H	53H	52H	51H	50H
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	47H	46H	45H	44H	43H	42H	41H	40H
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	37H	36H	35H	34H	33H	32H	31H	30H
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	27H	26H	25H	24H	23H	22H	21H	20H
23H	1FH	1EH	2DH	1CH	1BH	1AH	19H	18H
22H	17H	16H	15H	14H	13H	12H	11H	10H
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	07H	06H	05H	04H	03H	02H	01H	00H

3．数据缓冲区

MCS-51 单片机内部 RAM 中 30H～7FH 单元只能作为数据缓单元使用，暂时保存输入的数据或运算的结果。只有 52 子系列才有高端的 128B 的 RAM 单元，地址编号为 80H～FFH，只能采用间接寻址方式对其访问。

1.3.3 特殊功能寄存器

在 MCS-51 系列单片机内部的 I/O 口锁存器、定时器/计数器、串行口、中断等各种控制寄存器和状态寄存器都作为特殊功能寄存器（SFR），它们离散分布在 80H～0FFH 地址空间中，即每一个 SFR 均有一个存储单元映像地址，在这些映像地址中，只要是 8 的整数倍的 SFR 都可以按位寻址，有的还有位名称，如表 1.8 所示。

ACC 是累加器，它是运算器最重要的工作寄存器，用于存放参加运算的操作数据和运算结果。在指令系统中助记符 A 表示累加器。

B 寄存器是运算器中的一个工作寄存器，在乘法和除法运算中存放操作数据和运算结果，在其他运算中，可以作为一个中间结果寄存器使用。

SP 是 8 位的堆栈指针寄存器，数据进入堆栈前 SP 值加 1，数据弹出堆栈后 SP 减 1，复位后为 07H。若不对 SP 重新设置，则堆栈从 08H 单元开始。

DPTR 为 16 位的数据指针寄存器，它由 DPH 和 DPL 组成，一般作为访问外部数据存储器或 I/O 端口的地址指针寄存器使用，保存一个 16 位的地址，CPU 对 DPTR 操作也可以对高位字节 DPH 和低位字节 DPL 单独进行。

其他的特殊功能寄存器将在后续章节中作详细介绍。

表 1.8　特殊功能寄存器表

专用寄存器名称	符　　号	地址	位地址与位名称							
			D7	D6	D5	D4	D3	D2	D1	D0
P0 口	P0	80H	87H	86H	85H	84H	83H	82H	81H	80H
堆栈指针	SP	81H								
数据指针低字节	DPL	82H								
数据指针高字节	DPH	83H								
定时器/计数器控制	TCON	88H	TF1 8FH	TR1 8EH	TF0 8DH	TR0 8CH	IE1 8BH	IT1 8AH	IE0 89H	IT0 88H
定时器/计数器方式控制	TMOD	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0
定时器/计数器 0 低字节	TL0	8AH								
定时器/计数器 1 低字节	TL1	8BH								
定时器/计数器 0 高字节	TH0	8CH								
定时器/计数器 1 高字节	TH1	8DH								
P1 口	P1	90H	97H	96H	95H	94H	93H	92H	91H	90H
电源控制	PCON	97H	SMOD	—	—	—	GF1	GF0	PD	IDL
串行控制	SCON	98H	SM0 9FH	SM1 9EH	SM2 9DH	REN 9CH	TB8 9BH	RB8 9AH	TI 99H	RI 98H
串行数据缓冲器	SBUF	99H								
P2 口	P2	A0H	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H
中断允许控制	IE	A8H	EA AFH	—	ET2 ADH	ES ACH	ET1 ABH	EX1 AAH	ET0 A9H	EX0 A8H
P3 口	P3	B0H	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H
中断优先级控制	IP	B8H	—	—	PT2 BDH	PS BCH	PT1 BBH	PX1 BAH	PT0 B9H	PX0 B8H

续表

专用寄存器名称	符　　号	地址	位地址与位名称							
			D7	D6	D5	D4	D3	D2	D1	D0
定时器/计数器 2 控制	T2CON*	C8H	TF2 CFH	EXF2 CEH	RCLK CDH	TCLK CCH	EXEN2 CBH	TR2 CAH	C/T2 C9H	CP/RL2 C8H
定时器/计数器 2 自动重装低字节	RLDL*	CAH								
定时器/计数器 2 自动重装高字节	RLDH*	CBH								
定时器/计数器 2 低字节	TL2*	CCH								
定时器/计数器 2 高字节	TH2*	CDH								
程序状态字	PSW	D0H	C D7H	AC D7H	F0 D5H	RS1 D4H	RS0 D3H	OV D2H	—	P D0H
累加器	ACC	E0H	E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H
B 寄存器	B	F0H	F7H	F6H	F5H	F4H	F3H	F2H	F1H	F0H

注：表中带*的 SFR 只有 52 子系列单片机才有。

1.3.4　片外数据存储器

前面已经提到，MCS-51 系列单片机是通过 P0、P2 口线形成对外的地址总线，所以片外的数据存储器空间最大也是 64KB，单元地址范围为 0000H～FFFFH。但由于 MCS-51 单片机对扩展 RAM 和 I/O 端口统一编址，所以只要外部增加一些外围芯片，片外的 RAM 空间就会随之减少，而 CPU 对扩展的 RAM 和 I/O 端口的操作功能是相同的。

值得注意的是，MCS-51 单片机的程序存储器空间和外部数据存储器空间的都是 64KB，且地址范围都是 0000H～FFFFH，但两个空间是逻辑独立的，访问的形式和指令是不同的。

1.4　MSC-51 系列单片机的基本I/O口

MCS-51 单片机有 4 个双向 8 位并行 I/O 口 P0、P1、P2 和 P3，每一个 I/O 口的结构和使用方法有所不同。

1.4.1　P0 口的结构与特性

P0 口是一个多功能的 8 位口，可以字节访问也可以位访问，其字节访问地址为 80H，位访问地址为 80H～87H，P0 口的位结构原理图如图 1.8 所示。

P0 口中多路开关有两个输入，一是地址/数据输出，二是锁存器输出 $\overline{Q}$。多路开关的输出用于控制 T2 的导通和截止，多路开关的切换信号由内部控制信号控制。

P0 口的输出上拉电路导通和截止受内部控制信号和地址/数据信号共同来控制。当控制信号为 1 时，多路开关接通地址/数据信号，地址/数据输出线置 1 时，控制上拉电路的与门输出为 1，上拉管 T1 导通，同时地址/数据通过反相器输出 0，下拉管 T2 截止，A 点电位上拉为 1，地址/数据输出线为 1。当地址/数据输出线置 0 时，控制上拉电路的与门输出为 0，上拉管 T1 截止，同时地址/数据通过反相器输出 1，下拉管 T2 导通，A 点电位下拉为 0，地址/数据输出线为 0。

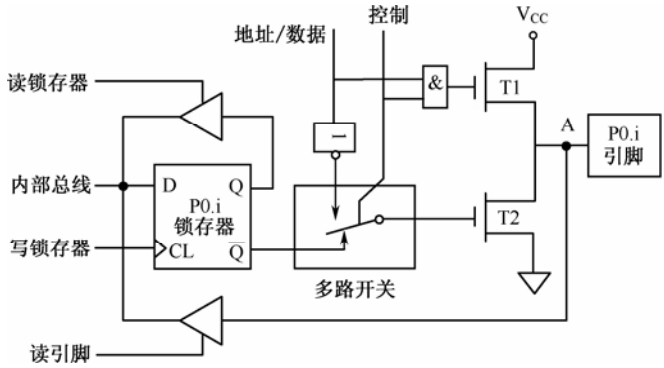


图 1.8　P0 口位结构原理图

当 P0 口控制信号置 0 时，多路开关接通输出锁存器 $\overline{Q}$ 端，此时 P0 口作为一般 I/O 口使用。由于内部控制信号为 0，与门关闭，上拉管 T1 截止，P0 口的输出电路为漏极开路输出，故需要外接上拉电阻才能正常工作。又因为输出锁存器的 $\overline{Q}$ 端引至下拉管 T2 的栅极，因此 P0 口的输出状态由下拉电路决定。

P0 口作输出口时，若 P0.i 输出 1，输出锁存器的 Q 端为 1，而 $\overline{Q}$ 端为 0，下拉管 T2 截止，此时 P0.i 为漏极开路输出，外接上拉电阻则输出为 1。若 P0.i 输出为 0，输出锁存器的 Q 端为 0，$\overline{Q}$ 端为 1，下拉管 T2 导通，此时 P0.i 被下拉为低电平，输出为 0。

P0 口作输入口时，为了使 P0.i 能正确读入引脚信号，必须先使 P0.i 置 1，下拉管 T2 截止，P0.i 处于悬浮状态，A 点的电平完全由外部电路来决定，通过读输入缓冲器就将 P0.i 上的电平信号读入 CPU。

P0 口有较强的输出驱动能力，可以直接驱动 8 个 LSTTL 门，在大多数场合中都能满足应用要求，而不必增加驱动电路。

1.4.2　P1 口的结构与特性

P1 口是一个 8 位口，可以按字节访问也可以按位访问，其字节访问地址为 90H，位访问地址为 90H～97H，P1 口的位结构原理图如图 1.9 所示。

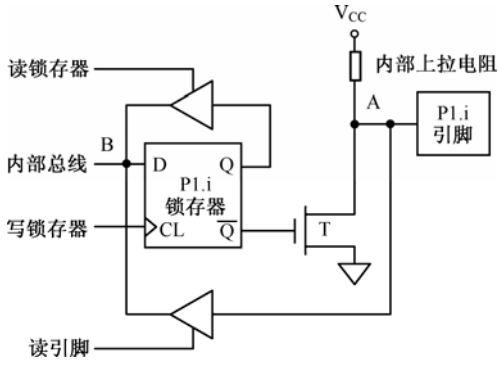


图 1.9　P1 口位结构原理图

P1 口包含输出锁存器、引脚输入缓冲器、锁存器输入缓冲器、下拉管 T 以及上拉电阻组成。

P1 口作输出口时，若 P1.i 输出 0，输出锁存器 Q=0，$\overline{Q}$ =1，下拉管 T 导通，A 点被下拉

为低电平，即输出 0。若 P1.i 输出 1，输出锁存器 Q=1，$\overline{Q}$=0，下拉管 T 截止，A 点处于高电平，即输出 1。

P1 口作输入口时，必须先向 P1.i 输出高电平，使下拉管处于截止状态。此时 A 点的电平高低完全由 P1.i 引脚上的电平决定，读取引脚缓冲器就可实现引脚信号输入。

由此可知，写输出锁存器没有任何条件，在进行输入时，须先将锁存器置 1，是一个“准”双向口。由于 P1 口没有高阻悬浮状态，它只有输入和输出两种状态。P1 口具有 4 个 LSTTL 门的驱动能力。

需要特别指出的是，若在输入操作之前不将输出锁存器置 1，如果 A 点为低电平，则引脚上的任何信号都被拉为低电平，外设信号无法实现输入。更为严重的是，当 A 点为低电平，而引脚为高电平，则这个高电平被下拉管 T 强制拉为低电平，将可能由于大电流而烧毁下拉管 T。

在 52 子系列中，P1.0 和 P1.1 除作为一般 I/O 口外，还具有第二功能，即

P1.0——定时器/计数器 2 的外部事件计数输入端；

P1.1——定时器/计数器 2 的捕获/重装控制端。

这时，P1.0 和 P1.1 的结构与 P3 口相似。

1.4.3 P2 口的结构与特性

P2 口是一个多功能 8 位口，可以按字节访问也可以按位访问，其字节访问地址为 A0H，位访问地址为 A0H～A7H，P2 口的位结构原理图如图 1.10 所示。

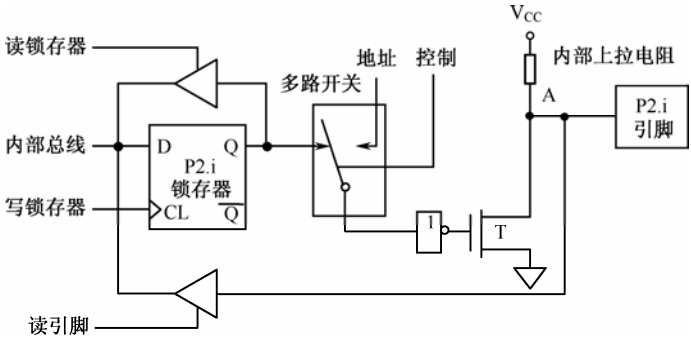


图 1.10 P2 口位结构原理图

从图 1.10 不难看出，P2 口与 P1 口比较，其结构基本相同，不同的是，在 P2 口中增加了一个多路开关和反相器。多路开关的输入有两个：一个是输出锁存器的输出端 Q，另一个是地址寄存器的高 8 位。多路开关的切换由内部控制信号控制。

P2 口作为一般 I/O 口时与 P1 口一样，也是一个准双向口，驱动能力也只有 4 个 LSTTL 门，其输入/输出过程和特性在此不再赘述。作为地址输出时，输出高 8 位地址信号，这个地址信号来自 PC 或 DPTR 的高 8 位。

在使用 P2 口时，需要注意的是：P2 口输出地址时其内部具有锁存功能，无须外接锁存器；当 P2 口作为地址使用后，就不能作为一般 I/O 口使用。

1.4.4 P3 口的结构与特性

P3 口也是一个多功能 8 位口，可以按字节访问也可以按位访问，其字节访问地址为 B0H，

位访问地址为 B0H～B7H，P3 口的位结构原理图如图 1.11 所示。

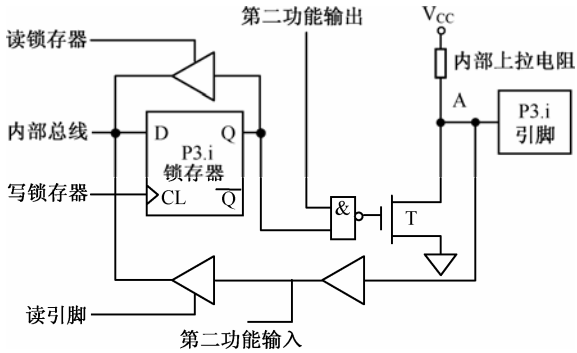


图 1.11　P3 口位结构原理图

从 P3 口的结构图中可以看出，它与 P1 口位结构基本相同，相比于 P1 口增加了一个与非门和一个输入缓冲器。与非门有两个输入端：一个为输出锁存器的 Q 端，另一个为第二功能输出。从 P3.i 输入有两个缓冲器，第二功能输入取自第一个缓冲器输出端；I/O 口的通用输入信号取自第二个缓冲器的输出端。

当第二功能输出置 1 时，输出锁存器的输出可以顺利通过到引脚 P3.i，其工作状况与 P1 口相类似，这时 P3 口的工作状态为 I/O 口，具有准双向口性质。当输出锁存器置 1 时，第二功能输出可以顺利通过引脚 P3.i，而作为第二功能输入的引脚，由于下拉管 T 截止，引脚电平可以通过缓冲器进入。

P3 口作为第二功能，各位有不同的功能，如表 1.3 所示。在大多数应用时，P3 口往往使用其第二功能（如读/写、中断等），但没有被使用的引脚仍然可以作为一般 I/O 线来使用。P3 口也具有 4 个 LSTTL 门的驱动能力。

1.5　MCS-51 系列单片机的工作方式

MCS-51 单片机的工作方式有：复位方式、程序执行方式、掉电和低功耗方式、编程方式、校验与加密方式等。复位方式在前面已经讲过，而程序执行方式是在复位后，单片机从 0000H 地址开始运行程序的方式，是单片机最基本的工作方式。由于“绿色电子”是电子产品的发展趋势，下面仅对掉电和低功耗方式作简单介绍。

MCS-51 系列的 CHMOS 型单片机运行时功耗低，而且还提供两种节电工作方式：待机方式（等待方式）和掉电方式（停机方式），以进一步降低功耗，它们特别适用于电源功耗要求很低的应用场合。这类应用系统往往是直流供电或停电时依靠备用电源供电，以维持系统的持续工作。CHMOS 型单片机的工作电源和后备电源加在同一个引脚 VCC，正常工作时电流为 11～20mA，空闲状态时为 1.7～5mA，掉电状态时为 5～50μA。空闲方式和掉电方式的内部控制电路如图 1.12 所示。在空闲方式中，振荡器保持工作，时钟脉冲继续输出到中断、串行口、定时器等功能部件，使它们继续工作，但时钟脉冲不再送到 CPU，因而 CPU 停止工作。在掉电方式中，振荡器工作停止，单片机内部所有的功能部件都停止工作。

CHMOS 型单片机的节电方式是由特殊功能寄存器 PCON 控制的。PCON 的格式如下：

SMOD	—	—	—	GF1	GF0	PD	IDL

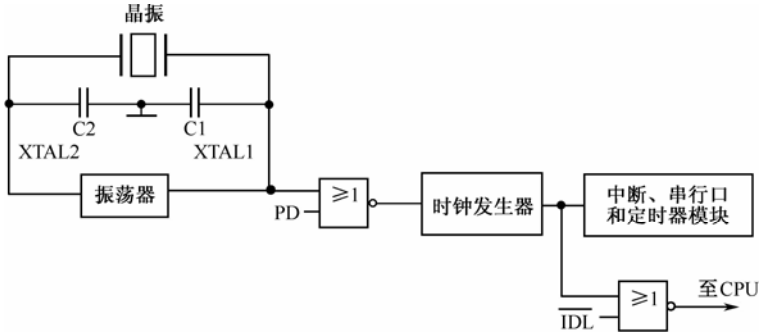


图 1.12　待机方式和掉电方式的内部控制电路图

其中，SMOD 为串口波特率倍率控制位；

GF1 为通用标志位；

GF0 为通用标志位；

PD 为掉电方式控制位，置 1 后使单片机进入掉电方式；

IDL 为空闲方式控制位，置 1 后使单片机进入空闲方式；

PCON.4～PCON.6 为保留位，对于 HMOS 型单片机仅 SMOD 位有效。对于 CHMOS 型单片机，当 IDL 和 PD 同时为 1 时，单片机进入掉电方式。

1. 待机方式

利用软件置 IDL 位为 1，单片机就进入待机工作方式。当单片机进入待机方式后，振荡器仍然工作，并提供给中断逻辑、串行口和定时器/计数器等，而不提供给 CPU，因此 CPU 不能工作。堆栈指针 SP、程序计数器 PC、程序状态字寄存器 PSW、累加器 ACC、内部数据存储器 RAM 和其他特殊功能寄存器内部维持不变，引脚状态保持进入空闲方式时的状态，ALE 和 $\overline{\text{PSEN}}$ 保持逻辑高电平。

单片机进入空闲方式后，有两种方法可以使单片机退出待机工作状态，即

（1）利用中断退出待机工作状态。即在空闲节电状态产生一个有效的中断请求，由内部电路对 IDL 进行清“0”，中止空闲方式，CPU 响应中断，执行相应的中断服务程序，中断处理结束后，从激活待机方式指令的下一条指令开始继续执行程序。

（2）硬件复位退出待机工作状态。即在待机状态对单片机实现有效的复位操作，从而使单片机进行初始化退出待机状态。

2. 掉电方式

利用软件置 PD 位为 1，单片机就进入掉电工作方式。在掉电工作方式下，CPU、中断、定时器/计数器、串行口等都停止工作。

一旦单片机进入掉电工作状态，唯一使单片机退出掉电方式的方法是硬件复位。单片机复位后对 PD 位进行初始化，将其清“0”，同时特殊功能寄存器的内容也被初始化，但 RAM 单元的内容保持不变。

在设计系统时，为了尽可能地降低系统的功耗，以延长电池的使用寿命。一方面系统中尽可能选用 CMOS 器件，另一方面在不影响系统性能的前提下降低时钟频率，还有就是充分利用 CHMOS 型单片机的节电工作方式。

本 章 小 结

单片机是嵌入式控制器的主要芯片，MCS-51 系列单片机及其兼容产品是我国单片机应用的主流产品，同时单片机技术的发展方兴未艾，出现了各具特色的单片机，如 Motorola 公司的系列单片机、Microchip 公司的 PIC 单片机、TI 公司的 MSP430 单片机等。学习 MCS-51 单片机具有代表性和实际意义。本章要求：了解 MCS-51 单片机的内部编程结构，这是应用单片机的基本前提；掌握独具特色的 MCS-51 单片机存储器组织及 I/O 口组织、内部存储单元、特殊功能寄存器的特性与特点，这是本章的一个重要内容；掌握单片机的复位方式、复位电路及复位状态；掌握 MCS-51 单片机基本 I/O 口的特性；掌握振荡周期、状态周期、机器周期和指令周期的基本概念；了解单片机节电工作方式。

习 题 1

1.1 什么是单片机？单片机的发展趋势如何？

1.2 单片机与一般计算机在结构上有什么不同？有何特点？

1.3 请上网搜索 5 个你认为办得最好的单片机专业网站。

1.4 MCS-51 系列单片机由哪些单元组成？其各自的功能有哪些？

1.5 MCS-51 单片机中的运算器和控制器各包括哪些功能部件？

1.6 MCS-51 单片机片内存储器组织有什么特点？

1.7 单片机 64KB 的程序存储器空间和数据存储器空间是如何区分的？内部 RAM 单元地址和位地址重叠，在操作时又是如何区分的？

1.8 程序计数器 PC 是多少位？单片机复位后其初始值为多少？其值说明了什么？

1.9 程序状态字寄存器 PSW 的各位的符号是什么？它们的含义是什么？

1.10 什么是时钟周期、状态周期和机器周期？三者的关系是什么？

1.11 如何通过对程序状态字寄存器 PSW 的相关位的操作选择当前工作寄存器区？

1.12 P0～P3 口各有哪些功能？为什么称其为准双向口？各口的驱动能力怎样？

1.13 P0 口作为一般 I/O 口时，为什么需要外接上拉电阻？

1.14 构成典型系统时，如何形成系统的地址总线、数据总线和控制总线？

1.15 CS-51 单片机的程序存储器和数据存储器的最大寻址空间为多少？

第 2 章　MCS-51 系列单片机的指令系统及汇编语言程序设计

本章主要内容

MCS-51 单片机指令系统。汇编语言的基本格式。程序结构及其设计方法。典型应用程序设计。

指令是 CPU 按照人们的意图来完成某种操作的命令。一台计算机全部指令集合称为指令系统。计算机的基本功能是执行程序，执行不同的程序就能完成不同的任务，而程序就是各种指令的有序组合。

2.1　指令格式与寻址方式

2.1.1　指令格式

为方便理解，请读者先观察下面左侧的程序片段（右侧是汇编后的机器码）。

	程 序 片 段		机器码
	MOV　SCON，#00H　；SCON←00H，串口工作于方式 0	75	98　00
	MOV　SBUF，A　；将 SBUF←（ACC）		F5　99
WAIT0：JC	TI，WAIT0　；发送完一个字节吗？	20	99　FD
	CLR　TI　；清除 TI 中断标志		C2　99

从上面的程序片段可以看出，在编写、阅读和修改 MCS-51 单片机程序时，采用汇编语言格式。在计算机执行程序时，则采用机器语言形式。MCS-51 汇编语言指令格式由以下几个部分组成：

[标号]:　　操作码　　[操作数]　　；[注释]

↓　　↓　　↙ ↘　　↓

WAIT0 :　　JC　　TI,　WAIT0　　；发送完一个字节吗？

标号：又称为指令地址符号。对标号有如下规定：

（1）一般由 1～8 个 ASCII 字符组成，以字母开头，其余字符可以是字母、数字和其他特定字符。

（2）不能使用助记符、伪指令或者寄存器的符号名称做标号。

（3）与操作码之间用冒号分开，带方括号表示为可选项。

（4）一个标号在程序中具有唯一性，但可以在指令中被多次引用。

操作码：是由助记符表示的字符串，它规定了指令要完成的具体操作。

操作数：是指参加操作的数据或数据的地址。操作数与操作码之间必须用空格分隔，如

有多个操作数需用逗号分隔。操作数项带有方括号，说明不是所有指令都具有操作数。

注释：是为该条指令所做的说明，以便于阅读，中英文不限，用“;”与指令分开，注释是每条指令的可选项。

指令在经过汇编之后形成机器码，根据机器码所占字节多少，把指令分成单字节、双字节、3 字节指令。比如“CLR　TI”这条指令就是双字节指令。

单片机在执行不同的指令时消耗的时间是不完全一样的，根据其消耗的机器周期数的不同又分为单周期、双周期和 4 周期指令。

在附录 B 中我们给出了 MCS-51 单片机的指令速查表，通过该表可以快捷地查出各条指令的作用、字节数、周期数以及对 PSW 状态位的影响。在学习过程中，读者要学会使用指令速查表。在指令的描述中经常用到一些特殊符号，介绍如下：

Rn：　表示当前寄存器区的 8 个工作寄存器 R0～R7（*n*=0～7）。

Ri：　表示当前寄存器区中的 R0 或 R1，可作地址指针即间址寄存器（i=0、1）。

@：　为间址寄存器或基址寄存器的前缀。如@Ri、@A+PC、@A+DPTR 等。

direct：　表示 8 位内部数据存储器单元的地址。它可以是内部 RAM 的单元地址 0～127、特殊功能寄存器 SFR 的地址（128～255）或名称，如 I/O 端口、控制寄存器、状态寄存器等。

#data：　表示包含在指令中的单字节（8bit）立即数。如果用十六进制表示，加后缀字母“H”，数据范围 00～0FFH，不得以字母开头；如果用十进制表示无须任何后缀，但必须在 0～255 之间。

#data16：　表示包含在指令中的双字节（16bit）立即数。

addrl6：　表示 16bit 目的地址。用于 LCALL 和 LJMP 指令中，目的地址范围是从 0000H～FFFFH 的整个 64KB 存储器地址空间。

addr11：　表示 11bit 目的地址。用于 ACALL 和 AJMP 的指令中，目的地址必须和下一条指令第一个字节同处一页。

rel：　表示 8bit 带符号的相对偏移量。用于 SJMP 和所有的条件转移指令中。偏移量相对于下一条指令的第一个字节计算，在−128～+127 范围内取值。

DPTR：　为数据指针，可用作 16bit 的地址寄存器。

bit：　表示内部可寻址位或特殊功能寄存器中的直接寻址位。

/：　加在位操数的前面，表示对该位进行非运算。

(x)：　某寄存器或地址单元的内容。

((x))：　有 x 间接寻址的单元中的内容。

←：　表示将箭头右边的内容传送至箭头的左边。

$：　当前指令的地址。

为了更清楚了解 MCS-51 单片机的指令系统，读者应该认识一下 MCS-51 单片机的寻址方式。

2.1.2　MCS-51 单片机寻址方式

所谓寻址方式就是计算机获取操作数的方法和途径，寻址方式越多，表示计算机的指令越丰富，操作越灵活。MCS-51 单片机一共有 7 种寻址方式，分别介绍如下。

1. 立即寻址

例：MOV　SCON，#00H　；将立即数 00H 送给 SCON，即 SCON←00H

在上面这条指令中，带下画线的操作数（在传送指令中称源操作数）称为立即数，直接在指令中给出，它是指令代码一部分，存放在程序存储器内的一个常数，这种寻址方式就称为立即寻址。

注意：立即数的前面必有“#”标志。

2. 直接寻址

例：MOV　A，30H　；将 30H 单元的内容给累加器 A，即(A)←(30H)

在上面这条指令中，带下画线的操作数（源操作数）是直接给出内部 RAM 中某个单元的地址或名称（特殊功能寄存器），这种寻址方式称为直接寻址。

直接寻址方式的寻址范围包括：

（1）内部 RAM 的 128 个单元。

（2）特殊功能寄存器。特殊功能寄存器除了以单元地址的形式给出，还可以用寄存器符号的形式给出。例如指令：

```
MOV　　A，80H
```

表示把 P0 口（字节地址为 80H）的内容传送给 A。也可以写为：

```
MOV　　A，P0
```

应当说明的是，直接寻址方式是对所有特殊功能寄存器读/写的唯一寻址方式。

3. 寄存器寻址

例：MOV　A，R7　；将通用寄存器 R7 中的数送给 A，即(A)←(R7)

在上面这条指令中，带下画线的操作数（源操作数）存放在当前工作寄存器组中某个寄存器中，这种寻址方式称为寄存器寻址。

寄存器寻址方式的寻址范围包括：

（1）4 组通用工作寄存器共 32 个工作寄存器。但只能寻址当前工作寄存器区的 8 个工作寄存器，因此指令中的寄存器名称只能是 R0～R7。

（2）部分特殊功能寄存器，如累加器 A、寄存器 B 以及数据指针寄存器 DPTR 等。

4. 寄存器间接寻址

```
例：MOV　R0，#3FH　　；(R0)←3FH，立即寻址方式
　　MOV　A，@R0　　；(A)←((R0))，即：(A)←(3FH)
```

在上面的第 2 条指令中，带下画线的操作数不是(R0)=3FH，而是以(R0)为地址，间接找到内部 RAM 中的 3FH 单元，将(3FH)中的数送给 A，这种寻址方式称为寄存器间接寻址。

当访问片内 RAM 或片外的低 256 字节 RAM 空间时，可用 R0 或 R1 作为间址寄存器；如果访问片外整个 64KB 的 RAM 地址空间时，可用 DPTR 作间址寄存器。例如：

```
MOV　DPTR，#0CFA0H　　；将 16 位数 CFA0H 赋给数据指针寄存器 DPTR
MOVX　A，@DPTR　　；将片外 RAM 或接口 CFA0H 单元内的数送给 A
```

5. 变址寻址

例：MOVC　A，@A+DPTR　　；将地址为(A)+(DPTR)的 ROM 单元中的数送给 A

在上面这条指令中，带下画线的操作数是以(DPTR)为基址，(A)为变址，将两者之和作为地址，找出该地址对应的 ROM 单元，再将该单元的数送给 A，这种寻址方式称为变址寻址方式。对该寻址方式做如下说明：

（1）该寻址方式只能对程序存储器进行寻址，寻址范围可达 64KB。

（2）该寻址方式的指令只有 3 条：

```
MOVC    A，@A+DPTR
MOVC    A，@A+PC
JMP     A，@A+DPTR
```

其中，前两条指令为读程序存储器指令，最后一条指令为无条件转移指令。

6. 相对寻址

```
例：CJNE        A，#80H，BUDENG     ；如(A)≠80H，跳转到 BUDENG
    SETB        FLAG0               ；(A)=80H, 置标志 FLAG0
    ……
BUDENG：  JNC   DAYU                ；如(A)＞80H，跳转到 DAYU
    ……
DAYU：    SETB  FLAG1               ；置标志 FLAG1
    ……
```

在上面这个程序片段中，带下画线的操作数指明了程序跳转的目的地址，即 PC 的当前值加上偏移量就构成程序转移的目的地址，这个目的地址可用如下公式表示：

目的地址=转移指令所在地址+转移指令的字节数+地址偏移量

其中偏移量是一个带符号的 8 位二进制补码数，其表示范围为−128～+127。上例中假设“CJNE A，#80H，BUDENG”(3 字节指令)在 ROM 中的存放位置是 2000H、2001H 和 2002H 单元，下一条语句“JNC　DAYU”的起始地址即 PC 就是 2003H，那么，“CJNE　A，#80H，BUDENG”的目标地址 BUDENG 的地址范围就是 1F83H(2003H−128)～2082H（2003H+127)。目标地址不得超出这个范围，否则程序出错。

相对寻址方式是为解决程序转移而专门设置的，被转移指令所采用。在指令表中，凡用 rel 表示的操作数都是相对寻址。

7. 位寻址

```
例：MOV     C，30H    ；将地址为 30H 的位的状态赋给进位位 Cy
    SETB    P1.0      ；将 P1 口（对应特殊功能寄存器 P1）的 P1.0 位置 1
```

在上面这两条指令中，带下画线的操作数不是某个字节地址或某个字节数，而是内部 RAM 中某个可直接寻址位的地址或名称，是针对位（bit）进行的操作，这种寻址方式称为位寻址。在指令表中，凡用 bit 和 C 做操作数的指令都属此类，它主要用在位操作指令和部分条件转移指令中。

位寻址方式的寻址范围包括以下两种情况：

（1）内部 RAM 中的位寻址区。单元地址为 20H～2FH，共 16 个单元，128 个位，位地址为 00H～7FH。

（2）特殊功能寄存器中的可寻址位。可供位寻址的特殊功能寄存器有 11 个，共有 88 个位，其中 5 位没有定义，所以有可寻址位 83 位。

以上介绍了 MCS-51 指令系统的 7 种寻址方式，概括起来如表 2.1 所示。

表 2.1　7 种寻址方式及其寻址空间

序号	寻 址 方 式	寻 址 空 间
1	立即寻址	程序存储器
2	直接寻址	片内 RAM 低 128 字节、特殊功能寄存器
3	寄存器寻址	4 组通用寄存器组、部分特殊功能寄存器
4	寄存器间接寻址	片内 RAM、片外数据存储器
5	变址寻址	程序存储器
6	相对寻址	程序存储器
7	位寻址	内部 RAM20H～2FH 单元的 128 个可寻址位、特殊功能寄存器的 83 个可寻址位

请读者思考，请说出以下几条指令分别属于哪种寻址方式：

① MOV　A，R0　　② MOV　C，20H

③ ADD　A，R1　　④ MOV　A，@R0

⑤ MOVC　A，@A+PC　　⑥ CJNE　A，#B，0FFFFH

⑦ ANL　A，00H　　⑧ MOV　A，#0FH

⑨ MOVX　A，@R0　　⑩ MOV　10H，20H

2.2　MCS-51 单片机指令系统

MCS-51 单片机指令系统有 5 类共 111 条指令，按照功能分类，可以分为以下五大类：

（1）数据传送类（29 条）。

（2）算术运算类（24 条）。

（3）逻辑运算类（24 条）。

（4）控制转移类（17 条）。

（5）位操作类（17 条）。

本节将向读者介绍各种指令的功能和用法。

2.2.1　数据传送和交换类指令

在 MCS-51 单片机中，数据传送是最基本和最主要的操作。MCS-51 单片机的数据传送指令共有 29 条。数据传送操作可以在片内 RAM 和 SFR 内进行，也可以在累加器 A 和片外存储器之间进行。指令中必须指定传送数据的源地址和目的地址，以便机器执行指令时把源地址中内容传送到目的地址中，但不改变源地址中的内容。在这类指令中，除以累加器 A 为目的操作数，指令会对特殊寄存器 PSW 中的奇偶标志位 P 有影响外，其余指令执行时均不会影响任何标志位。

1. 内部数据传送指令

这类指令的基本格式为：

MOV　目的地址，源操作数

内部数据传送是在单片机内部 128 字节 RAM 和 SFR 范围内进行的，如图 2.1 所示。

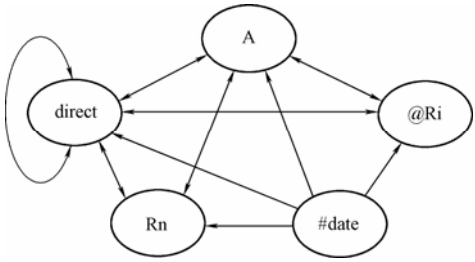


图 2.1　MCS-51 单片机内部数据传送示意图

（1）以累加器 A 为目的的传送指令：

```
MOV    A，#data          ；(A)←data
MOV    A，direct         ；(A)←(direct)
MOV    A，Rn             ；(A)←(Rn)
MOV    A，@Ri            ；(A)←((Ri))
```

（2）以通用寄存器 Rn 为目的的传送指令：

```
MOV    Rn，A             ；(Rn)←(A)
MOV    Rn，direct        ；(Rn)←(direct)
MOV    Rn，#data         ；(Rn)←data
```

（3）以直接地址为目的的传送指令：

```
MOV    direct，A         ；(direct)←(A)
MOV    direct，Rn        ；(direct)←(Rn)
MOV    direct1，direct2  ；(direct1)←(direct2)
MOV    direct，@Ri       ；(direct)←((Ri))
MOV    direct，#data     ；(direct)←data
```

（4）以寄存器间接地址为目的的传送指令：

```
MOV    @Ri，A            ；((Ri))←(A)
MOV    @Ri，direct       ；((Ri))←(direct)
MOV    @Ri，#data        ；((Ri))←data
```

例 2.1　设计一个最简单的程序，让图 2.2 所示电路中 VD0～VD4 发光，VD5～VD7 熄灭。

解：

```
ORG    0000H             ；定位伪指令，参见 2.3.1 节
MOV    P1，#1FH          ；P1←00011111B
END                      ；结束伪指令，参见 2.3.1 节
```

例 2.2　分析在执行完以下指令后，R0 和内部 RAM 中 40H 单元的内容。

```
ORG      0000H
MOV      R0，#40H        ；R0←40H
MOV      40H，#10        ；(40H)←10
MOV      @R0，#00H       ；((R0))←00H
END
```

解：(R0)=40H，(40H)=00H

例 2.3　设计一个简单的程序，将内部 RAM 中 30H 单元的数据通过图 2.2 中的 VD0～VD7 显示出来。

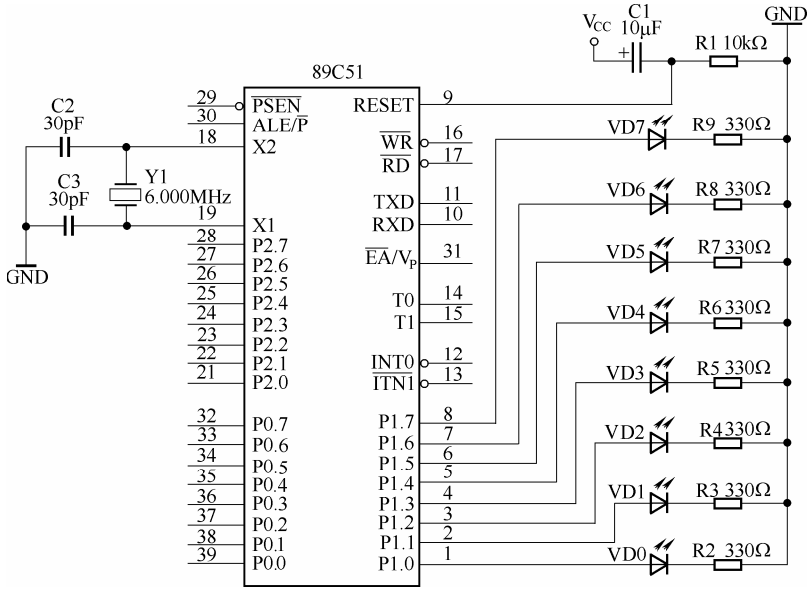


图 2.2　P1 口应用电路

解：ORG　0000H

MOV　P1，30H　；P1←(30H)

END

例 2.4　分析在执行完如下程序后，各空格的数值分别为多少？

ORG　0000H

MOV　R0，#40H

MOV　40H，#30H

MOV　A，@R0

MOV　R0，A

END

(R0)=_____　(40H)=_____　(A)=_____

解：(R0)=30H　(40H)=30H　(A)=30H

请读者思考，如何设计一个简单的程序，将寄存器 R6 和 R7 中的数互换。

2. 数据指针赋值指令（16bit数据传送指令）

当要对片外 RAM、I/O 接口进行访问，或进行查表操作时，一般要先给数据指针 DPTR 赋值，此时应用下条指令：

MOV　DPTR，#data16　；将 16bit 二进制立即数赋给数据指针 DPTR

例如，将数据指针指向片外 RAM 的 2000H 单元：

MOV　DPTR，#2000H

3. 片外数据传送指令

当把一个数送到单片机外部扩展 RAM 的某个单元或外部扩展 I/O 口某个端口，或者从该单元、端口将数据取回到单片机内部的时候，就要用到片外数据传送指令，它们如下：

```
MOVX  A，@Ri          ；(A)← ((Ri)) 片外
MOVX  A，@DPTR        ；(A)← ((DPTR)) 片外
MOVX  @Ri，A          ；((Ri)) 片外←(A)
MOVX  @DPTR，  A ；((DPTR)) 片外←(A)
```

如果以 Ri 作指针，只能访问 00～FFH 地址段的外部 RAM 或 I/O 端口。如果以 DPTR 作指针，则能访问 0000～FFFFH 地址段的任何外部 RAM 单元或 I/O 端口。

例 2.5　假如在图 2.2 的基础上为 89C51 扩展一片地址为 0000～07FFH 的外部 RAM，设计一个程序，将外部 RAM 中 0100H 单元的数送到 P1 口，通过 VD7～VD0 显示出来。

解：凡涉及到对片外 RAM 的访问，都必须用外部数据传送指令。由于本例中外部 RAM 的地址已经超出 00～FFH 范围，所以不能用 Ri 作指针，只能用 DPTR 作指针，即：

```
ORG     0000H
MOV     DPTR，#0100H  ；首先将要访问的地址赋给数据指针 DPTR
MOVX    A，@DPTR      ；取回数据，存放在累加器 A 中
MOV     P1，A         ；数据送到 P1 口显示
END
```

请读者设计一段程序，将外部 RAM 中 0010H 单元中的数送给 0100H 单元，并将外部 RAM 中 0010H 单元和 0200H 单元中的数互换。

4. ROM数据访问指令（查表指令）

在 MCS-51 单片机程序设计中，经常涉及到要查阅 ROM 中存放的数据，即查表。为此，MCS-51 单片机指令系统中专门设置了两条 ROM 数据访问指令，它们分别是：

```
MOVC  A，@A+DPTR      ；(A)←((A)+(DPTR))ROM
MOVC  A，@A+PC        ；(PC)←(PC)+1，(A)←((A)+(PC))ROM
```

注意，因为程序运行中 ROM 只读，所以 ROM 地址只能做源地址，不能做目的地址。

例 2.6　当（A）=30H 时，执行地址为 1000H 处的指令 MOVC　A，@A+PC 后，累加器中的内容是什么？

解：由于该指令占用一个字节，下一条指令的地址为 1001H，所以((A)+(PC))=1031H，结果是将 1031H 单元的内容送到累加器 A 中。

请读者思考：MCS-51 单片机的运算功能是比较差的，采用何种方法可以让它快速便捷地运算函数：y=sin(x)？

5. 堆栈操作指令

在 MCS-51 内部 RAM 中可以设定一个后进先出（LIFO）的区域，称为堆栈。在特殊功能寄存器中有一个堆栈指针 SP，它指定堆栈栈顶的位置。堆栈操作在子程序调用与返回、中断服务程序进入和退出过程中非常重要，它往往用来临时存放一些重要的数据。堆栈操作有进栈和出栈两种，因此在指令系统中相应有两条指令分别是：

```
PUSH   direct         ；(SP)←(SP)+1，(SP)←(direct)
                      ；堆栈指针先加 1，将数据压入栈顶
POP    direct         ；(direct)←(SP)，(SP)←(SP)－1
                      ；将数据从栈顶弹出存入 direct，SP 再减 1
```

堆栈在使用过程中，如果要让数据正确返回到原地址，应遵循“先入后出、后入先出”

的原则。同时注意指令的操作数是 direct 直接地址，不能是间接地址、通用寄存器或其他形式。

例 2.7 分析下列程序运行结束后，累加器 ACC 和特殊功能寄存器 TH0 的值。

```
ORG     0000H
MOV     SP，#60H        ；将栈底设置为内部 RAM 的 60H 单元
MOV     A，#10H         ；累加器赋值 10H
MOV     TH0，#20H       ；特殊功能寄存器 TH0 赋值 20H
PUSH    ACC             ；将累加器中的数压入堆栈，注意不能写成“PUSH  A”
PUSH    TH0             ；再压入 TH0 中的数
POP     ACC             ；先弹出栈顶的数给 ACC，注意不能写成“POP  A”
POP     TH0             ；再弹出栈顶的数给 TH0
END
```

解： 上列程序中的堆栈操作过程可以用图 2.3 描述。

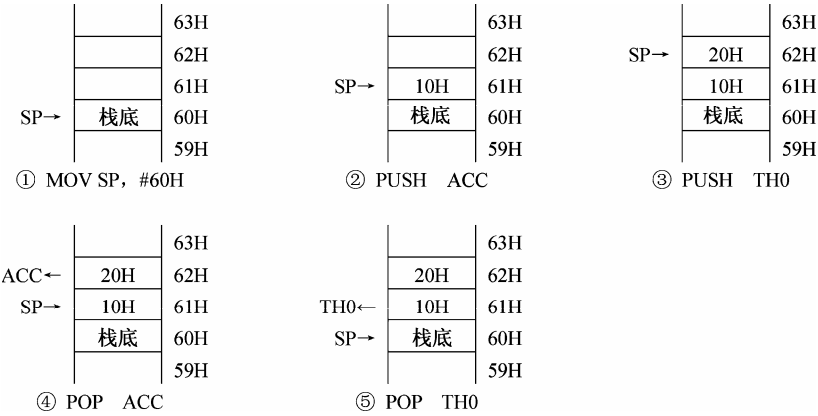


图 2.3 堆栈的操作过程

通过图 2.3，我们很容易发现，程序结束后，ACC 和 TH0 中的数发生了交换，(ACC)=20H，(TH0)=10H。为什么出现这种结果？原因就在于程序没有遵循“先入后出、后入先出”原则。如果我们想让堆栈中的数据正确返回原地址，程序应改为：

```
……
PUSH    ACC         ；先压入
PUSH    TH0         ；后压入
POP     TH0         ；先弹出
POP     ACC         ；后弹出
……
```

6. 数据交换指令

数据交换指令共有 5 条，分别是：

（1）整字节（8bit）交换指令：

```
XCH   A，   Rn          ；A 和 Rn 中的数互换
XCH   A，   direct      ；A 和 direct 单元中的数互换
XCH   A，   @Ri         ；A 和 Ri 间址单元中的数互换
```

（2）半字节交换指令：

XCHD　A，　@Ri　　；A 的低 4 位和 Ri 间址单元的低 4 位互换，高 4 位不动

（3）累加器高低半字节交换指令：

SWAP　A　　；A 的高 4 位(D7～D4)和低 4 位（D3～D0）互换

例 2.8　假设(A)=79H，(R0)=20H，(20H)=85H，试分析分别执行以下 4 条指令后，相关单元的数分别应该是多少？

① XCH　A，R0　　(A)=_____　(R0)=_____

② XCH　A，@R0　　(A)=_____　(R0)=_____　(20H)=_____

③ XCHD　A，@R0　　(A)=_____　(R0)=_____　(20H)=_____

④ SWAP　A　　(A)=_____

解：① XCH　A，R0

数据交换前：(A)=79H，(R0)=20H，交换后：(A)= 20H　(R0)= 79H

② XCH　A，@R0

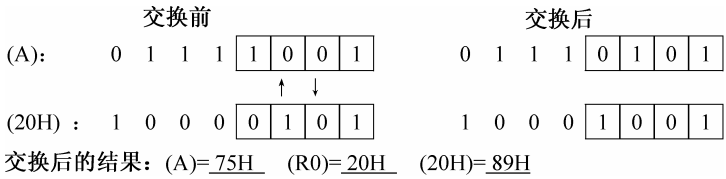
该指令是将 A 和(R0)即 20H 单元的数进行互换，R0 的数不受影响。

交换前：(A)=79H，(R0)=20H，(20H)=85H

交换后：(A)= 85H　(R0)= 20H　(20H)= 79H

③ XCHD　A，@R0

该指令是将 A 和(R0)即 20H 单元的低 4 位进行互换，交换过程如下：



交换后的结果：(A)= 75H　(R0)= 20H　(20H)= 89H

④ SWAP　A

该指令将累加器 A 的高 4 位(D7～D4)和低 4 位(D3～D0)互换，过程如下：

交换前：(A)=79H= 0 1 1 1　1 0 0 1 B

交换后：(A)= 1 0 0 1　0 1 1 1　B=97H

请读者思考：如果要将内部 RAM 中 20H 单元的数和外部 RAM 中 20H 单元的数进行交换，你能以哪些方式实现该操作，编写出相应程序，看看谁最简单。

2.2.2　算术运算类指令

MCS-5l 有比较丰富的算术运算指令，共 24 条，可以分为加、减、乘、除和十进制调整五类。除加 1 和减 1 指令外，其余指令会影响 PSW 的 P、OV、AC、Cy 标志位。

1. 加法指令

加法指令共有 13 条，由不带进位位 Cy 加法、带 Cy 加法和加 l 指令等三种组成。

（1）不带 Cy 加法指令（4 条）：

```
ADD   A,   Rn        ; (A)←(A)+(Rn)
ADD   A,   direct    ; (A)←(A)+(direct)
ADD   A,   @Ri       ; (A)←(A)+((Ri))
ADD   A,   #data     ; (A)←(A)+data
```

这 4 条指令的功能是把 A 中的数与源操作数相加，其结果仍存在 A 中。相加过程中若 D3 位有进位则将辅助进位标志 AC 置位，若 D7 位有进位，则置位进位标志 Cy，否则清 0。

两个无符号数相加时，若 Cy 置位，说明和大于 255(FFH)。两个有符号数相加时，当 D6 位或 D7 位之中只有一位产生进位时，溢出标志位 OV 置位，说明和产生了溢出，即大于+127 或小于－128，结果错误。

例 2.9 分析在执行如下加法指令后，PSW 寄存器中各位的状态。

```
MOV  A，#48H
ADD  A，#5CH
……
```

解：为便于理解，我们分析一下加法运算的过程：

	D7	D6	D5	D4	D3	D2	D1	D0	
	0	1	0	0	1	0	0	0	48H
+	0	1	0	1	1	1	0	0	5CH
	0	1	0	1	0	0	1	0	0	A4H
	Cy	C6			AC				

C6 代表 D6 位向 D7 位的进位。

显然，本例中：AC=1、C6=1、Cy=0、OV= Cy⊕C6=1，由于(A)有奇数个 1，故 P=1，PSW 中其他各位不受影响。加法结束后，(A)=A4H。

在上面这个例子中，如果两个操作数 48H 和 5CH 是无符号数，则运算结果是正确的。假设它们是带符号数，则可以根据 PSW 中的 OV=1 便可知晓加法运算中产生了溢出，运算的结果是错误的。

请读者试一试，分析在执行如下加法指令后，PSW 寄存器中各位的状态。

```
MOV  A，#79H
ADD  A，#97H
```

（2）带进位加法指令（4 条）：

```
ADDC  A，Rn        ；(A)←(A)+(Rn)+Cy
ADDC  A，direct    ；(A)←(A)+(direct)+Cy
ADDC  A，@Ri       ；(A)←(A)+((Ri))+Cy
ADDC  A，#data     ；(A)←(A)+ data +Cy
```

这 4 条指令的功能是把源操作数所指示的内容和 A 中的内容以及进位标志 Cy 相加，结果存入 A 中。运算结果对 PSW 中相关位的影响和不带进位加法指令相同。这类指令主要用在多字节加法运算中。

例 2.10 有两个双字节无符号数 x 和 y，x 存放在内部 RAM 的 31H(高 8 位)和 30H(低 8 位)单元中，y 存放在内部 RAM 的 33H（高 8 位）和 32H（低 8 位）单元中，设计一段程序计算 x+y，将和存放在 35H（高 8 位）和 34H（低 8 位）中。

解：先将 x 和 y 各自的低 8 位进行不带进位相加，在把高 8 位进行带进位相加即可。

```
MOV   A，  30H     ；先把 x 的低 8 位送给累加器 A
ADD   A，  32H     ；与 y 的低 8 位相加，和在 A 中，进位在 Cy
MOV   34H，A       ；将低 8 位相加的和送给 34H
MOV   A，  31H     ；先把 x 的高 8 位送给累加器 A
ADDC  A，  33H     ；带上低 8 位进位，高 8 位相加，Cy 存高 8 位的进位
MOV   35H，A       ；将高 8 位相加的和送给 35H
```

在这个例子中，真实的和应该是由 Cy 、35H 单元、34H 单元 3 个部分组成。请读者思考一下，如果要将 Cy 的状态存放到 36H 单元中，如何设计程序？

（3）加 1 指令（5 条）：

```
INC   A          ；(A)←(A)+1
INC   Rn         ；(Rn)←(Rn)+1
INC   @Ri        ；((Ri))←((Ri))+1
INC   direct     ；(direct)←(direct)+1
INC   DPTR       ；(DPTR)←(DPTR)+1
```

这组指令不影响 PSW 的任何位，只是将各自对应单元内的数加 1。

例如，已知(A)=0FH、(40H)=10H、(R0)=40H、(SP)=60H、(DPTR)=1FFH，如执行以下指令：

```
INC   A
INC   @R0
INC   SP
INC   DPTR
```

其结果为：(A)=10H、(40H)=11H、(R0)=40H、(SP)=61H、(DPTR)=200H。

2. 减法指令

（1）带进位减法指令（4 条）：

```
SUBB  A，  Rn          ；(A)←(A)－(Rn)－Cy
SUBB  A，direct        ；(A)←(A)－(direct)－Cy
SUBB  A，@Ri           ；(A)←(A)－((Ri))－Cy
SUBB  A，#data         ；(A)←(A)－data－Cy
```

这 4 条指令的功能是把 A 中的内容减去源操作数所指出的内容和进位位 Cy，差存入 A 中。如果被减数≥减数，进位位 Cy 被清 0；如果被减数＜减数，低字节需向高字节借位，因此进位位 Cy 将被置 1。这 4 条指令对 OV、AC、和 P 等 3 个状态位的影响方式与加法指令雷同，只不过是借位而非进位。

例 2.11　分析在执行如下减法指令后，PSW 寄存器中各位的状态。

```
SETB  C          ；将 Cy 置位，即 Cy←1
MOV   A，#48H
SUBB  A，#5CH
```

解：为便于理解，我们分析一下减法运算的过程：

```
    0   1   0   0   1   0   0   0      (A)= 48H
    0   1   0   1   1   1   0   0      data= 5CH
－                          1          Cy=1
-------------------------------------------------
    1   1   1   1   0   1   0   1   1  (A)= EBH
    Cy  C6          AC
```

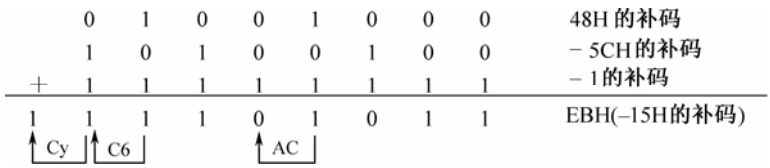
C6 代表 D6 位向 D7 位的借位。

在减法过程中：Cy、C6、AC 都发生借位，故它们全为 1，OV= Cy ⊕ C6=0，P=0。

如果程序中 48H 和 5CH 两个数被看作无符号数，则计算结果错误，因为 Cy=1。如果 48H 和 5CH 两个数被看作有符号数，则因 OV=0，无溢出发生，结果正确，因为 EBH 正好是−15H

的补码。

如果你觉得上面的减法过程不好理解，你可以把减法运算变成被减数补码和减数补码相加。实际上计算机内部就是这么做的。过程如下：



结果和前面的方法一样。

注意：在 MCS-51 单片机的指令系统中，没有不带 Cy 的减法指令。

（2）减 1 指令（4 条）：

```
DEC     A                   ；(A)←(A)－1
DEC     direct              ；(direct)←(direct)－1
DEC     Rn                  ；(Rn)←(Rn)－1
DEC     @Ri                 ；((Ri))←((Ri))－1
```

这组指令可以使指令中源地址所指 RAM 单元中内容减 l。和加 1 指令一样，减 1 指令也不影响 PSW 标志位状态，只是第一条减 1 指令对奇偶校验标志位 P 有影响。

例如，假设(A)=10H、(20H)=1AH、(R0)=30H、(30H)=1FH、(R3)=00H，如执行以下程序段：

```
DEC     A
DEC     20H
DEC     R3
DEC     @R0
```

程序执行后，(A)=0FH、(20H)=19H、(R0)=30H、(30H)=1EH、(R3)=0FFH。

注意，在 MCS-51 单片机中，不能对数据指针 DPTR 直接进行减 1 操作。

3. 乘除指令

MCS-51 单片机指令系统中有单字节乘法和除法指令各一条，这两条指令执行的时间都是 4 机器周期，是所有指令中，执行时间最长的。

（1）乘法指令：

```
MUL     AB    ；(B) (A)←(A)×(B)
```

指令功能是把累加器 A 和特殊功能寄存器 B 中两个 8 位无符号整数相乘，并把积的高 8 位字节存入 B 寄存器，低 8 位字节存入累加器 A。本指令执行过程中将对 PSW 的 Cy、OV 和 P 三个标志位产生影响。其中，Cy 位被清 0；奇偶校验标志位 P 仍由累加器 A 中 l 的奇偶性确定；OV 标志位用来表示积的大小，若乘积超过 0FFH，即(B)≠0，则 OV=1，否则 OV=0。

例 2.12　假设已知(A)=15H，(B)=2AH，分析在执行指令：

```
MUL     AB
```

后，(A)、(B)、Cy、OV 和 P 分别是多少？

解：因为　(A)×(B)=15H×2AH=372H＞0FFH

所以　指令执行后，(B)=03H、(A)=72H、OV=1、P=0、Cy=0。

例 2.13　设计一段程序，将内部 RAM 中 43H 和 42H 两单元内的数相乘，乘积的低 8

位存入 41H，高 8 位存入 40H。

解：由于本例中几个 RAM 单元的地址相邻，可以考虑用间址寻址方式。程序如下：

```
ORG    0000H
MOV    R0，#43H
MOV    A，@R0        ；(A)←(43H)
DEC    R0
MOV    B，@R0        ；(B)←(42H)
MUL    AB
DEC    R0
MOV    @R0，A        ；(41H)←(A)
DEC    R0
MOV    @R0，B        ；(40H)←(B)
END
```

请读者思考：如果把内部 RAM 中 41H(高 8 位)单元、40H(低 8 位)单元存放的一个双字节 16 位无符号数 x，和 42H 单元存放的 8 位无符号数 y 相乘，再把乘积从高到低依次存入 43H、44H 和 45H 单元。程序如何设计？

（2）除法指令：

```
DIV    AB    ；(A)÷(B)，商存入 A，余数存入 B
```

指令功能是把累加器 A 中的 8 位无符号整数除以寄存器 B 中的 8 位无符号整数，商的整数部分存入累加器 A 中，余数保留在 B 中。除法指令执行后，将使 Cy=0，P 标志由 A 决定，只是对溢出标志位 OV 的影响不一样。在除法指令执行过程中，如 B 寄存器中的除数为 0，则 OV 被自动置 1，表示除法没有意义；其余情况下，OV 均被清 0。

例 2.14　假设已知(A)=0C8H，(B)=06H，分析在执行指令

```
DIV    AB
```

后，(A)、(B)、Cy、OV 和 P 分别是多少？

解：因为　除数(B) ≠0，且 (A)÷(B)，商=21H，余数=02H

所以　指令执行后，(A)=21H、(B)=02H、OV=0、P=0、Cy=0。

4. 十进制调整指令

在单片机中，如果要进行十进制（BCD 码）的加运算，如何才能得到正确结果呢？看看下面这个例子。

假设有一个 BCD 码的十进制数 9 放在 A 中，即(A)=$00001001_{(BCD)}$，同时有另一个 BCD 码的十进制数 6 放在 30H 单元中，即(30H)=$00000110_{(BCD)}$。如果执行指令“ADD　A，30H”，即把这两个十进制数相加，和应该还是一个十进制数，正确的结果应该是(A)=15D=$00010101_{(BCD)}$。但是单片机执行的情况是这样的：

```
      0  0  0  0  1  0  0  1
+     0  0  0  0  0  1  1  0
-------------------------------
      0  0  0  0  1  1  1  1
```

运算结果不正确，(A)≠$00010101_{(BCD)}$，产生错误的原因是 BCD 码虽然是 4 位二进制编码，但只用了其中的 10 个编码，其余 6 个（1010、1011、1100、1101、1110、1111）为无效编码。凡结果进入或者跳过无效编码区时，其结果就是错误的。在 BCD 码的加法运算中，1 位 BCD

码加法运算出错的情况有以下两种：

（1）相加结果大于 9，已经进入无效编码区。

（2）相加结果有进位，已经跳过无效编码区。

如果要得到正确结果，我们需要在加法指令后增加一条十进制调整指令，对(A)进行调整。调整的具体操作是：

（1）相加结果的低 4 位大于 9 或产生 AC 进位，则低 4 位加 6。

（2）相加结果的高 4 位大于 9 或产生 Cy 进位，则高 4 位加 6。

这条指令就是十进制调整指令：

```
DA   A
```

例 2.15 （A）=56H，(10H)=67H，两数均为压缩的 BCD 数，进行 BCD 数加法后，执行下列指令后，结果是什么？

```
MOV     A，#56H
ADD     A，10H
DA      A
```

解：

```
      0  1  0  1  0  1  1  0
  +   0  1  1  0  0  1  1  1
  ---------------------------
      1  0  1  1  1  1  0  1
  +   0  1  1  0  0  1  1  0   ← 十进制调整，高、低 4 位分别加 6
  ---------------------------
    1 0  0  1  0  0  0  1  1
```

结果为（A）=23H，Cy=1。

由上可见，56+67=123，结果正确。

DA A 是一条专用指令，只用于对 BCD 码十进制数加法运算结果进行修正。请读者思考：BCD 减法该怎样调整呢？

2.2.3 逻辑运算指令

在程序设计中，经常会遇到对数的某些位进行清 0、置 1 或移位等逻辑处理，这时就会用到逻辑运算指令，该类指令共有 24 条，除以累加器 A 为目标寄存器指令外，其余指令均不会影响 PSW 的任何标志位。

1. 逻辑与运算指令

逻辑与运算是按位进行的，用符号“∧”表示，共有如下 6 条指令：

```
ANL   A，Rn          ；(A)←(A)∧(Rn)
ANL   A，direct      ；(A)←(A)∧(direct)
ANL   A，@Ri         ；(A)←(A)∧((Ri))
ANL   A，#data       ；(A)←(A)∧data
ANL   direct，A      ；(direct)←(A)∧(direct)
ANL   direct，#data  ；(direct)←(direct)∧data
```

例 2.16 假设已知(R1)=39H，执行以下指令后，(R1)为多少？

```
MOV    A，R1
ANL     A，#0CDH
MOV    R1，A
```

解：本例程序执行后，实际上(R1)=39H∧0CDH，过程如下：

```
      0  0  1  1  1  0  0  1
∧     1  1  0  0  1  1  0  1
-------------------------------
      0  0  0  0  1  0  0  1
```

程序执行结束后，(R1)=09H。

用逻辑与指令可以方便地把目标操作数中的某些位清 0（称为屏蔽），而其他位不变。

请读者思考：设计一个程序片段，让图 2.2 所示电路中 VD3～VD0 全部熄灭，VD7～VD4 保持不变。

2. 逻辑或运算指令

逻辑或运算也是按位进行的，用符号“∨”表示，共有如下 6 条指令：

```
ORL   A，Rn              ；(A)←(A)∨(Rn)
ORL   A，direct          ；(A)←(A)∨(direct)
ORL   A，@Ri             ；(A)←(A)∨((Ri))
ORL   A，#data           ；(A)←(A)∨data
ORL   direct，A          ；(direct)←(A)∨(direct)
ORL   direct，#data      ；(direct)←(direct)∨data
```

用逻辑或指令可以方便地把目标操作数中的某些位置 1，而其他位不变，也称为屏蔽。

例如，假设已知(30H)=56H，则执行“ORL　30H，#0FH”后，(30H)=5FH。

请读者思考：设计一个程序片段，让图 2.2 所示电路中 VD7～VD3 全部发光，VD2～VD0 保持不变。

3. 逻辑异或运算指令

逻辑异或运算形式和前两种一样，只是运算方式变成异或，用符号“⊕”表示：

```
XRL   A，Rn              ；A←(A)⊕(Rn)
XRL   A，direct          ；(A)←(A)⊕(direct)
XRL   A，@Ri             ；(A)←(A)⊕((Ri))
XRL   A，#data           ；(A)←(A)⊕data
XRL   direct，A          ；(direct)←(A)⊕(direct)
XRL   direct，#data      ；(direct)←(direct)⊕data
```

用逻辑异或指令可以十分方便地把目标操作数中的某些位取反，而其他位不变。

例 2.17　设计一段程序，将 P1 寄存器的 D7～D4 位取反，而 D3～D0 位不变。

解：这种部分位取反操作用异或操作是很容易完成的，用一条指令即可完成：

```
XRL   P1，#0F0H
```

验证：假设(P1)=97H，执行“XRL　P1，#0F0H”，即

```
      1  0  0  1  0  1  1  1        (P1)=97H
⊕     1  1  1  1  0  0  0  0        #0F0H
-------------------------------
      0  1  1  0  0  1  1  1        (P1)=67H
```

我们看到，P1 的高 4 位中，为 1 的位变成 0，为 0 的位变成 1，而 P1 的低 4 位维持不变。

请读者思考：假如例 2.17 中是要求 D5～D0 位取反，而 D7、D6 位不变，程序又如何设计？这其中有什么规律可循吗？

4. 累加器清 0 和取反指令

（1）累加器清 0 指令：

```
CLR    A                ；(A)←0
```

（2）累加器取反指令：

```
CPL    A                ；(A)←(A)‾
```

注意，累加器取反也是按位进行的。

例 2.18　设计一段程序，将内部 RAM 中 30H、31H 单元内的数进行同或运算，结果存入 33H 单元。

解：MCS-51 单片机没有同或指令，同或运算可以通过先进行异或再取反来实现。程序如下：

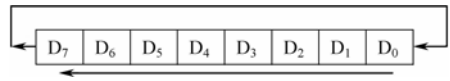
```
MOV    A，30H           ；(A)←(30H)
XRL    A，31H           ；(A)←(30H)⊕(31H)
CPL    A                ；(A)= (30H)⊙(31H)
MOV    33H，A
```

5. 累加器移位指令

就像数字电路里有移位寄存器一样，51 指令系统里有 4 条指令可以对累加器进行移位操作，解决移位和循环这类逻辑问题。它们分别是：

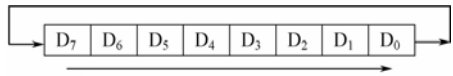
（1）不带进位 Cy 循环左移：

RL　A　；　D_{n+1}←D_n，D_0←D_7



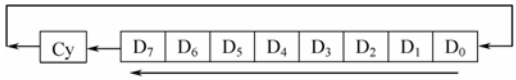
（2）不带进位 Cy 循环右移：

RR　A　；　D_{n+1}→D_n，D_0→D_7



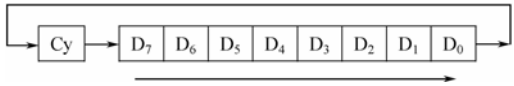
（3）带进位 Cy 循环左移：

RLC　A　；Cy←D7，Dn+1←Dn，D0←Cy



（4）带进位 Cy 循环右移：

RRC　A　；Cy→D_7，D_{n+1}→D_n，D_0→Cy



例 2.19　设计一个程序，让如图 2.2 所示电路中各发光二极管按如下规律循环发光（流水灯）。

VD_7	VD_6	VD_5	VD_4	VD_3	VD_2	VD_1	VD_0
*	*						
	*	*					
		*	*				
			*	*			
				*	*		
					*	*	
						*	*
*							*

* ：发光

空格：不发光

每0.5秒切换一次

解：很明显，各 LED 的显示规律是一个循环右移的过程，用循环指令来设计比较方便，下面是一个比较完整的例程：

```
        ORG     0000H
        MOV     A，#0C0H        ；置显示初值，让 VD7、VD6 发光
LOOP：  MOV     P1，A
        ACALL   DELAY           ；调用延时 0.5 秒子程序
        RR      A               ；A 循环右移 1 位，切换到下一组显示
        SJMP    LOOP        ；跳转到 LOOP 语句，形成循环
DELAY： ……                      ；延时子程序
        END
```

请读者思考：在例 2.19 中，如果改变发光二极管的循环方向，程序如何修改？如果要改变成每次是相邻的 4 个发光二极管发光，程序又如何修改？以此类推，你是否可以设计一个彩灯控制器了？

2.2.4 控制转移指令

在例 2.19 中，我们已经接触到两条控制转移指令了。实际上，在程序设计过程中，控制转移指令是非常重要的，没有这类指令，我们几乎不可能设计出一个有实用价值的程序。控制转移指令主要用来改变程序的执行顺序，比如条件分支、跳转和子程序调用等。MCS-51 单片机的控制转移类指令共 17 条。

1. 无条件转移指令

如例 2.19 中的指令“SJMP　LOOP”，当程序执行到这条语句时，无条件地跳转到指定的位置即标号为“LOOP”的目标程序语句，这就是一个典型的无条件转移。

在 MCS-51 单片机指令系统中，根据目标程序语句的位置不同有 4 种跳转方式。

（1）长转移指令（3 字节指令）：

```
LJMP    addr16          ；(PC)←addr16
```

指令中，addr16 往往是某条目标语句的标号，这条目标语句可以在 64KB ROM 空间的任何位置。换句话讲，用长转移指令可以跳转到程序的任何一条语句。

（2）绝对转移指令（双字节指令）：

```
AJMP    addr11              ；(PC)←(PC)+2，(PC)10~0←addr11
```

addr11 通常是目标语句的标号。

长转移是 3 字节指令，而绝对转移指令只有 2 字节。绝对转移指令的转移地址必须与 AJMP 下一条指令的第一个字节在同一 2KB 范围内，即转移的目标地址必须与 AJMP 下一条指令的高 5 位地址码 A15～A11 相同，否则汇编程序会提示相应的错误。

（3）短转移指令（双字节指令）：

```
SJMP    rel             ；(PC)←(PC)+2+ rel
```

这是一种相对寻址指令。在前面我们已经知道，相对寻址中－128≤rel≤+127。在短转移指令中，rel 通常是目标语句的标号。

短转移指令对跳转的范围也有严格的限制。它要求：

当前语句的 PC－126≤目标语句的 PC≤当前语句的 PC+129

如果目标语句超出这个范围，将产生汇编错误。

（4）变址寻址转移指令（单字节指令）：

```
JMP    @A+DPTR          ；(PC)←(A)+(DPTR)
```

这种转移的目的地址是由 A 的内容和 DPTR 的内容共同确定，即目的地址的(PC)=(A)+(DPTR)。这条指令主要用在多分支程序(散转程序)设计中，参见 2.3.2 节。

2. 条件转移指令

在程序设计中，经常遇到这样的问题：当满足条件时怎么样，不满足条件时又怎么样。实际上这就是一种条件转移，需要用到条件转移指令来完成。通常，执行条件转移指令时，满足条件就跳转到目标语句，不满足条件则顺序执行。

MCS-51 单片机提供了 3 种条件转移指令。它们的目标地址都由相对寻址方式确定，因此对目标地址的限制和短转移指令一样，这一点请读者一定要注意。

（1）累加器判 0 转移指令（双字节指令）：

```
JZ          rel   ；如果(A)=0，跳转到目标语句，否则顺序执行
JNZ         rel   ；如果(A)≠0，跳转到目标语句，否则顺序执行
```

rel 通常是某条目标语句的标号。

例 2.20　设计一段程序，实现功能：如果(SBUF)=00H，P1←01H，否则 P1←80H。

解：这是一种最典型的条件转移，条件是(SBUF)=0？程序如下：

```
        MOV     A,   SBUF
        JZ      NEXT        ；如果为 0，跳到 NEXT 语句
        MOV     P1,   #80H  ；否则 P1←80H
        ……
NEXT:   MOV     P1，#01H
        ……
```

（2）比较转移指令：比较转移指令是把两个数进行比较，如不相等则跳转到指定目标语句，否则程序顺序执行。其基本形式为：

```
CJNE  左操作数，右操作数，目标地址
```

这类指令属于 3 字节指令，共有 4 条：

```
CJNE  A，#data，rel          ；如果(A)≠data，则跳转到目标语句，否则程序顺序执行
CJNE  A，direct，rel         ；如果(A) ≠ (direct)，则跳转到目标语句，否则程序顺序执行
CJNE  Rn，#data，rel         ；如果(Rn)≠data，则跳转到目标语句，否则程序顺序执行
CJNE  @Ri，#data，rel        ；如果((Ri)) ≠data，则跳转到目标语句，否则程序顺序执行
```

由于这 4 条指令是 3 字节指令，所以对目标地址有如下要求：

当前语句的 PC－125≤目标语句的 PC≤当前语句的 PC+130

同时请读者注意，这 4 条指令会影响 PSW 标志位 Cy，具体为：

左操作数≥右操作数，则 Cy=0；左操作数＜右操作数，则 Cy=1

例 2.21　参照图 2.2，设计一段程序实现功能：将内部 RAM 30H 和 31H 两个单元的数进行比较，如果(30H)=(31H)，则让发光二极管 VD0 发光；如果(30H) ≠ (31H)，则让发光二极管 VD1 发光。

解：程序如下：

```
        ORG     0000H
        MOV     A，30H
```

```
                CJNE    A，31H，BUDENG
                MOV     P1，#01H            ；如(30H)=(31H)，让发光二极管 VD0 发光
                SJMP    JIESHU              ；跳转到结束
BUDENG：        MOV     P1，#02H            ；(30H)≠(31H)，让发光二极管 VD1 发光
                SJMP    JIESHU
JIESHU：        SJMP    $                   ；程序执行完，“原地踏步”
                END
```

在本例中，用到一条“原地踏步”指令，其目的在于让程序执行到此止步，便于观察输出结果。

（3）循环控制转移指令：

```
DJNZ    Rn，rel     ；(Rn)先减 1，如减 1 后(Rn)≠0，则跳转到目标语句
                    ；否则顺序执行
```

这条指令是双字节指令，对目标语句地址的限制和短转移指令相同。

```
DJNZ    direct，rel     ；(direct)先减 1，如减 1 后(direct)≠0，则跳转到目标语句
                        ；否则顺序执行
```

这是一条 3 字节指令，对目标语句地址的限制和比较转移指令相同。

例 2.22　设计一段程序，实现功能：将内部 RAM20H～2FH 单元全部清 0。

解：本例很明显是在重复同一个操作：先将 20H 单元清 0、再将 21H 单元清 0……，一直到把 2FH 单元清 0。像这种重复的操作，最好用循环程序来完成，程序如下：

```
                ORG     0000H
                MOV     R0，#20H
                MOV     R2，#10H            ；置循环次数
                CLR     A
LOOP：          MOV     @R0，A
                INC     R0
                DJNZ    R2，LOOP
                SJMP    $
                END
```

请读者思考：设计一段程序，实现功能：在外部 RAM 中的 1020H～103FH 单元内分别置数 00H、01H、02H～1FH。

3. 子程序调用和返回指令

凡学过程序语言的读者都知道，在程序设计过程中，为了缩短程序长度、增加程序的结构性和可读性，我们通常把一些反复执行的功能模块设计成子程序，比如延时子程序、显示子程序、键盘查询子程序等等。和子程序密切相关的就是子程序的调用和返回指令。MCS-51 单片机中共有 4 条指令与此相关。

（1）绝对调用指令：

```
ACALL    addr11 ；addr11 通常是子程序第 1 条语句的标号
```

与绝对转移指令类似，绝对调用指令对被调用子程序的位置 addr11 有很严格的限制，子程序只能在 2KB 范围内才能被调用。

为保证子程序调用结束后，程序能正确回到调用处，单片机在进入子程序前还有一项工作要完成：将调用指令的下一条语句的 PC 压入堆栈。一个子程序的绝对调用过程可以表示为：

①(PC)←(PC)+2

②(SP)←(SP)+1，　　$(SP)\leftarrow(PC)_{7\sim0}$

③(SP)←(SP)+1，　　$(SP)\leftarrow(PC)_{15\sim8}$

④$(PC)_{10\sim0}$←addr11

例 2.23　设（SP）=52H，标号 PRC=0312H，子程序 SUB1 起始地址为 0645H，执行指令

PRC：ACALL　SUB1

后，（SP）=_____，（53H）=_____，（54H）=_____，（PC）=_____。

解：（SP）= 54H ，（53H）= 14H ，（54H）= 03H ，（PC）= 0645H 。

（2）长调用指令：

LCALL　　addr16　；addr16 通常是子程序第 1 条语句的标号

与长转移指令类似，LCALL 指令可以调用 64KB ROM 空间中的任何子程序，不像绝对调用指令对子程序的地址有比较严格的限制。所以对初学者而言，如果你觉得 ACALL 不易掌握，不妨就用 LCALL 指令。

长调用指令是一条 3 字节指令。调用过程可表示为：

①(PC)←(PC)+3

②(SP)←(SP)+1，　　$(SP)\leftarrow(PC)_{7\sim0}$

③(SP)←(SP)+1，　　$(SP)\leftarrow(PC)_{15\sim8}$

④(PC)←addr16

例 2.24　设（SP）=52H，标号 PRC=105DH，子程序 SUB1 起始地址为 3000H，执行指令

PRC：　LCALL　SUB1

后，（SP）=_____，（53H）=_____，（54H）=_____，（PC）=_____。

解：（SP）= 54H ，（53H）= 60H ，（54H）= 10H ，（PC）= 3000H 。

（3）返回指令：单片机在执行完子程序或中断服务程序后，要返回到主程序或被中断的程序继续执行，这是通过返回指令来实现的。

RET　　；子程序返回

RETI　　；中断服务程序返回

无论是子程序返回指令，还是中断服务程序返回指令，都是从堆栈中取出断点地址，送给 PC，使程序从断点处继续执行，返回过程为：

①$(PC)_{15\sim8}$←(SP)，　　(SP)←(SP)-1

②$(PC)_{7\sim0}$←(SP)，　　(SP)←(SP)-1

必须提请注意的是：子程序返回指令和中断返回指令虽然其功能相同，但两者不能相互替代！子程序返回指令只能用在子程序末尾，而中断服务程序末尾只能使用中断返回指令。

4. 空操作指令

空操作指令格式如下：

NOP

空操作指令是一条特殊指令，单片机在执行该指令时不进行任何操作，只是消耗 1 个机器周期的时间，所以该指令常用于延时程序、软件陷阱程序等。

2.2.5　位操作类指令

MCS-51 硬件结构中有一个布尔处理器，因此设有一类专门处理布尔变量的指令，又称

为位操作指令，这类指令可以实现位传送、位逻辑运算、控制程序转移功能，是 MCS-51 单片机指令系统中非常重要而独具特色的一类指令，它使得面向自动控制的程序设计起来更加灵活，效率更高。MCS-51 单片机的位操作类指令共有 17 条。

1. 位传送指令

位传送指令就是内部 RAM 中的可寻址位和 Cy 之间的相互传送，共有两条：

```
MOV   C，bit          ；(Cy)←(bit)，bit 位的状态不变
MOV   bit，C          ；(bit)←(Cy)，Cy 位的状态不变
```

比如：

```
MOV    P1.0，C        ；将 Cy 位的状态传送给 P1 口的第 0 位
MOV    C，10H         ；将内部 RAM 中 10H 位(22H.0)的状态传送给 Cy
MOV    ACC.6，C       ；将 Cy 位的状态传送给累加器 ACC 的第 6 位
```

2. 位置位和复位指令

这种指令可以让指定位置 1 或清 0。

```
SETB   C              ；(Cy)←1
SETB   bit            ；(bit)←1
CLR    C              ；(Cy)←0
CLR    bit            ；(bit)←0
```

如果我们想让图 2.2 中的发光二极管 VD1 发光，可以用指令：

```
MOV    P1，   #02H
```

也可以用指令：

```
SETB   P1.1
```

显然用置位指令更灵活，因为它直接针对 P1.1，可以不影响 P1 口的其他位。

3. 位运算指令

位运算和数字电路里讲述的逻辑运算一样。MCS-51 单片机指令系统提供了位与、或、非 3 种最基本的逻辑运算，读者可以在此基础上构件更复杂的逻辑运算。

```
ANL    C，bit         ；(Cy)←(Cy(∧(bit)，Cy 位和 bit 位相与，结果赋给 Cy
ANL    C，/bit        ；(Cy)←(Cy)∧(/bit)，Cy 位和/bit 相与，结果赋给 Cy
ORL    C，bit         ；(Cy)←(Cy)∨(bit)，Cy 位和 bit 位相或，结果赋给 Cy
ORL    C，/bit        ；(Cy)←(Cy)∨(/bit)，Cy 位和/bit 相或，结果赋给 Cy
CPL    C              ；(Cy)←(/Cy)，Cy 位取反
CPL    bit            ；(bit)←(/bit)，bit 位取反
```

例 2.25　参照图 2.2，设计一段程序实现功能：如果 VD0 和 VD1 同时亮或同时灭则 VD7 亮，如果 VD0 和 VD1 一个亮一个灭则 VD7 灭。

解：很显然，本例是要求你直接针对 P1.0、P1.1、P1.7 三位进行位操作，P1.7 和 P1.0、P1.1 之间实际是一个同或关系：

$$P1.7=\overline{P1.0}\cdot\overline{P1.1}+P1.0\cdot P1.1$$

程序如下：

```
MOV    C，P1.0
ANL    C，P1.1        ；(Cy)←(P1.0)∧(P1.1)
```

```
MOV     00H，C          ；将 Cy 暂存入 00H 位
MOV     C，P1.0
CPL     C
ANL     C，/ P1.1       ；(Cy)←(P1.0 )∧(P1.1 )
ORL     C，00H          ；(Cy)←(P1.0 )∧(P1.1 )+ (P1.0)∧(P1.1)
MOV     P1.7，C
```

4. 位测试转移指令

在前面我们学习了控制转移指令，这类指令的判断条件是以字节为单位进行的。除此之外还有一种控制转移指令，其判断的条件是某可寻址位的状态，这就是位测试转移指令。

（1）以 Cy 位状态为条件的转移指令（双字节指令）：

```
JC      rel             ；如果(Cy)=1，跳转到目标语句，否则顺序执行
JNC     rel             ；如果(Cy)=0，跳转到目标语句，否则顺序执行
```

（2）以指定位状态为条件的转移指令（3 字节指令）：

```
JB      bit，rel        ；如果(bit)=1，跳转到目标语句，否则顺序执行
JNB     bit，rel        ；如果(bit)=0，跳转到目标语句，否则顺序执行
JBC     bit，rel        ；如果(bit)=1，跳转到目标语句，同时将 bit 位清 0；否则顺序执行
```

例 2.26 比较单片机内部 RAM 中 50H 和 52H 中的两个无符号数的大小，将大数存入 50H，小数存入 52H 中；若两数相等，则将片内 RAM 中的位地址 60H 置“1”。

解：本例首先要判断两个数是否相等，若不相等，再判断两个数的大小，因此要用两个条件转移指令，程序如下：

```
BIJIAO：    MOV     A，50H
            CJNE    A，52H，Q1      ；两数不相等，则转 Q1
            SETB    60H             ；两数相等，60H 置“1”
            SJMP    Q2
Q1：        JNC     Q2              ；(Cy)=0，(50H)>( 52H)，则转 Q2
            MOV     50H，52H        ；(50H)<( 52H)，两数交换
            MOV     52H，A
Q2：        SJMP    $
            END
```

2.3 汇编语言程序设计

程序是若干指令的有序集合，单片机的运行就是执行这一指令序列的过程，编写这一指令序列的过程称为程序设计。

MCS-51 单片机的程序设计主要采用两种语言，一种是汇编语言，另一种是高级语言（如 C51）。目前较多设计人员采用高级语言（C51）来完成单片机的应用程序设计，但是在对程序的空间和时间要求很高的场合，汇编语言仍是不可或缺的。汇编语言编写的程序具有效率高、占用存储空间少、运行速度快、实时性强的特点，对单片机的硬件资源操作直接方便、概念清晰，对于学习和掌握单片机的硬件结构极为有利。

本节将介绍基于汇编语言的程序设计的基本思路和方法，熟悉常规的程序结构。本书将

在第 8 章介绍 C51 程序设计语言及程序设计方法。

2.3.1 MCS-51 单片机汇编语言的伪指令

汇编语言源程序必须“汇编”成为机器代码后，才能运行。细心的读者可能早已发现，在前面的许多例程中出现了像“ORG 0000H”、“END”等指令，这些指令在汇编过程中，向汇编程序提供指示信息，告诉它如何完成汇编工作，称这些指令为伪指令。伪指令不属于 MCS-51 单片机指令系统中的指令，在汇编后不产生机器码，而只是在程序进行汇编时，对汇编程序进行控制和提供某些指示，比如：程序起止、表格存放位置等，汇编结束后，伪指令就不存在了。

MCS-51 单片机汇编语言程序设计中，常用的伪指令有以下 7 种：

1. ORG(Origin)定位伪指令

```
ORG     addr16
```

该伪指令用于规定其后面指令在程序存储器 ROM 中存放的起始地址。在汇编时，当汇编软件检测到该语句时，它就把该语句后面的所有指令或数据依次存入以 ORG 后面的 16 位地址或标号所指定的存储单元为首地址的 ROM 单元中。

2. END(End of assembly)结束汇编伪指令

END 伪指令称为结束汇编伪指令，用来指示源程序到此全部结束。在汇编时，当汇编软件检测到该语句时，它就确认汇编语言源程序已经到此为止，对 END 后面的指令都不予汇编。因此，一个源程序只能有一个 END 语句，而且必须放在整个程序末尾。

3. EQU(Equate)赋值伪指令

Equate 的中文意思是“使……和……相等”，所以该指令用于给变量或常量赋值。用 EQU 赋值的常量或变量，其值在整个程序中有效。

该指令的格式：

```
变量/常量名称    EQU    赋值项
```

赋值项可以是常数、地址、符号或表达式。赋值后的常量或变量既可以做地址使用，也可以作立即数使用。例如，

```
DUANKOU     EQU    P1                ；DUANKOU=P1
            MOV    DUANKOU，#00H     ；(P1)←00H
            ANL    A，DUANKOU        ；(A)←(A)∧(P1)
            ……
```

在上面这个例子中，如果我们想让 DUANKOU=P3，直接更改 EQU 后面的值即可。

4. DB(Define Byte)定义字节指令

该指令的功能是用于从指定的地址开始，在程序存储器的连续单元中定义字节数据，常用于在 ROM 中存放表格数据和字符串。

例 2.27 从 ROM 的 200H 单元开始存放 0～9 共十个数的共阴数码管七段显示代码，从 ROM 的 210H 单元开始存放字符串“how do you do”。

解：……

```
ORG   200H
DB   3FH，06H，5BH，4FH，66H   ；数之间用逗号分隔，跳行再用 DB 开头
DB   6DH，7DH，07H，7FH，6FH
ORG   210H
DB   ‘how do you do’
```

注意：定义字符串时要用单引号把字符串括起来。

5. DW(Define Word)定义数据字指令

该指令用于从指定地址开始，在程序存储器的连续单元中定义双字节的数据。例如，

```
ORG   1000H
DW    0011H，2233H  ；(1000H)=00H，(1001H)=11H，(1002H)=22H，(1003H)=33H
DW    ‘DO’，‘IT’    ；(1004H)=44H，44H 是 D 的 ASCII 码，以此类推：
                    ；(1005H)=4FH，(1006H)=49H，(1007H)=54H
```

如果定义数据，则必须是双字节；如果是定义字符串，则单括号内只能有两个字符。

6. DS(Define Storage)定义存储区指令

该指令用于从指定的 ROM 地址开始，保留指定数目的字节单元作为存储区，供程序运行使用。在汇编时，对这些单元不赋值。例如：

```
ORG   2000H
DS     10H    ；从 2000H 单元开始，保留 16 个连续的字节单元。
```

7. BIT位定义指令

指令的格式：

```
位名称   BIT   位地址
```

该语句的功能是把 BIT 右边的位地址赋给它左边的“位名称”。例如，

```
D0    BIT     P1.0
D1    BIT     P1.1
      SETB    D0      ；(P1.0) ←1
      CLR     D1      ；(P1.1) ←0
```

2.3.2 程序结构

尽管我们可以用汇编语言设计出千变万化的程序，但程序的基本结构只有三种：顺序结构、分支结构和循环结构。

1. 顺序结构

这是最简单的程序结构，其显著的特点是：程序中的语句由前向后顺序执行，直到最后一条指令，程序中没有任何条件判断语句。这种结构的程序通常只能完成一些简单的操作。

2. 分支结构

分支结构相对而言比顺序结构要复杂一点。分支程序是通过转移指令完成的，通常又有单分支和多分支两种结构。

（1）单分支程序。单分支程序都是使用条件转移指令实现的。

例 2.28　参照图 2.2，设计一段程序实现功能：如果(A)有奇数个 1，则让 P1 口所有发光二极管全部发光；如果全 0，则只让 VD3～VD0 发光；否则全灭。

解：根据题意，可以画出其流程图如图 2.4 所示，程序如下：

```
        ORG     0000H
        JB      P，JISHU
        JZ      QUANLING
        MOV     P1，#00H
        SJMP    $
JISHU:  MOV     P1，#0FFH
        SJMP    $
QUANLING: MOV   P1，#0FH
        SJMP    $
        END
```

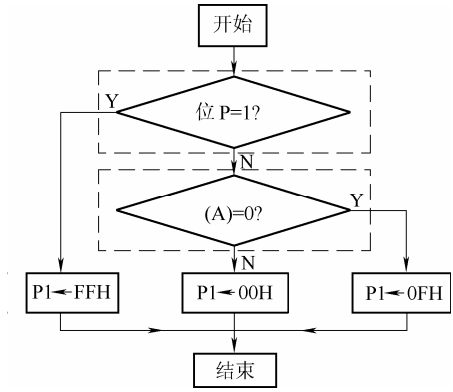


图 2.4　单分支结构

在图 2.4 中，每个虚线框内就是一个单分支，该例中一共有两个单分支。读者千万不要认为单分支就是程序中只能有一个分支结构。

MCS-51 单片机中，每条 JZ、JNZ、DJNZ、CJNE、JC、JNC、JB、JNB、JBC 等条件跳转指令就是一个单分支。

请读者思考：设计一段程序实现功能：如果(30H)=00H，调用子程序 SUB0；如果(30H)=08H，调用子程序 SUB1；如果(30H)=50H，调用子程序 SUB2。

（2）多分支程序。为便于读者理解多分支结构，请你先看下面这个例子。

例 2.29　在某单片机应用系统中，接有一键盘，键值（代表哪个键被按下）存放在内部 RAM 的 40H 单元内。设计一段程序实现功能：如果(40H)=00H，调用子程序 SUB1；如果(40H)=01H，调用子程序 SUB2；如果(40H)=02H，调用子程序 SUB3；如果(40H)=03H，调用子程序 SUB4；如果(40H)=04H，调用子程序 SUB5。

解：先画出本例的示意流程图，如图 2.5 所示。程序如下：

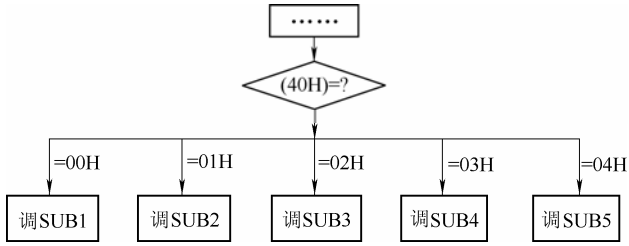


图 2.5　典型多分支结构

```
……
MOV     A，40H
MOV     DPTR，#TABLE
RL      A               ；A←(A)×2
ADD     A，40H          ；A←(40H)×3
JMP     @A+DPTR
```

```
        ……
TABLE:  LCALL   SUB1
        LCALL   SUB2
        LCALL   SUB3
        LCALL   SUB4
        LCALL   SUB5
        ……
```

从例 2.29 可以清楚地发现，多分支结构和单分支结构有比较明显的区别，它往往是根据输入条件或运算结果来确定转向相应的处理程序，这种结构通常又称为散转结构。MCS-51 单片机指令系统中利用 JMP @A+DPTR 指令可以方便地编制散转程序，由数据指针 DPTR 决定多分支转移程序表的首地址，累加器 A 的内容动态地选择相应的分支程序。

在例 2.29 中，由于 LCALL 指令是 3 字节指令，为保证 JMP 指令执行后能正确调用相应子程序，所以在 JMP 指令前要将(40H)乘 3 并存入 A 中。读者可以给 40H 带入数据验证一下。如果我们采用 ACALL 指令调用子程序，则在 JMP 指令前应将(40H)乘 2 并存入 A 中。为何如此，请读者思考。

例 2.29 示范了实现多分支程序的一种方法，实际上还有其他不同的方法也可以实现多分支程序，限于篇幅，不在赘述。

3. 循环结构

程序设计中，经常遇到要反复执行同一个程序段，这时可以采用循环结构的程序来完成，以缩短程序长度，节省存储空间。

循环程序一般由以下 4 部分组成：

（1）初始化部分。这是循环程序的准备部分，如给循环次数计数器、地址指针和某些变量赋初值。

（2）循环处理。为反复执行的程序段，是循环程序的主体部分，又称循环体。

（3）循环控制。在重复执行循环体的过程中，不断修改循环控制变量，为进入下一轮处理作准备，直到符合结束循环的条件为止。

（4）循环结束。分析及存放程序执行结果。

循环程序的基本结构一般有两种形式：

（1）先进入循环处理部分，再控制循环，如图 2.6（a）所示，这种结构至少执行一次循环体。

（2）先控制循环，根据判断结果决定是否进入循环处理部分，如图 2.6（b）所示。

循环次数的控制根据实际情况而定：如果循环次数已知，可以用计数器来控制；循环次数未知时，可以用某种条件进行控制。

例 2.30　设计一段程序实现功能：统计累加器 A 的 8 位数中一共有多少个 1，把结果存入 30H 单元。

解：要统计(A)中 1 的个数，可以采用这种思路：用 RLC 指令把 A 带上 Cy 循环左移，如果移入 Cy 的是 1，就让(30H)加 1；重复 8 次这个操作，不就可以统计出结果了吗？由此可以画出相应的流程图如图 2.7 所示。

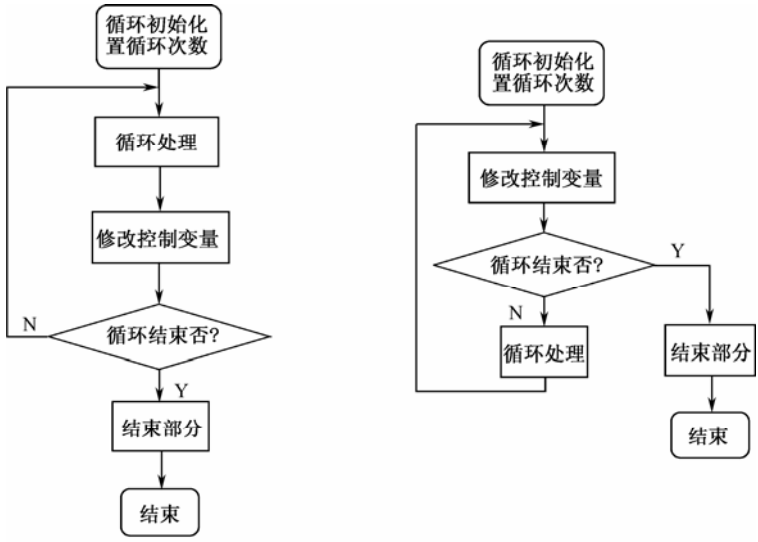


图 2.6　基本循环结构

程序如下：

```
        ……
        MOV    30H，#00H      ；循环初始化，计数单元清零
        MOV    R2，#08H       ；置循环次数
LOOP：  RLC    A
        JNC    NEXT
        INC    30H
NEXT：  DJNZ   R2，LOOP       ；判断循环是否结束
        ……
```

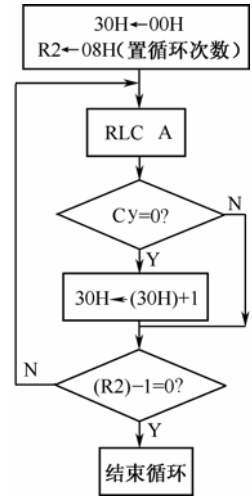


图 2.7　循环结构实例

例 2.31　设有一字符串，依次存放在单片机内部 RAM 40H 单元开始的连续单元中，该字符串以 0FH 为结束标志，试编写一段程序测试字符串长度。

解：本例中要测试字符串长度，需将 RAM 40H 开始的单元的内容依次与 0FH 比较，由于循环次数未知，为此需要预先设定一个长度计数器和字符串地址指针。如果比较结果不相等，则计数器计数一次，并继续比较下一个单元的内容；如果比较结果相等，则表示找到字符串结束标志，计数器的值即为字符串长度。程序如下：

```
        MOV    R0，#3FH
        MOV    R1，#0FFH
LOOP：  INC    R0
        INC    R1
        CJNE   @R0，#0FH，LOOP
        SJMP   $
```

以上循环程序例子都只包含一个循环程序，称为单重循环程序。如果一个循环程序中包含其他循环程序，则称为多重循环程序。软件延时程序是最常见的多重循环程序之一。

例 2.32　假设单片机晶振频率 f_{osc}=12MHz，编制用软件方法延时 0.1s 的程序。

解：软件延时时间与 MCS-51 单片机的指令周期以及晶振频率有关，如果 f_{osc}=12MHz，一个机器周期为 1μs。用双重循环方法编写程序如下：

```
DELAY2：    MOV   R3，  #0C8H        ；第①条语句，1 个机器周期
LOOP1：     MOV   R2，  #0F8H        ；第②条语句，1 个机器周期
            NOP                      ；第③条语句，1 个机器周期
LOOP2：     DJNZ  R2，LOOP2          ；第④条语句，2 个机器周期
            DJNZ  R3，LOOP1          ；第⑤条语句，2 个机器周期
            RET                      ；第⑥条语句，2 个机器周期
```

本例由两层循环构成，LOOP1 是外层循环，循环次数为 200 次；LOOP2 是内层循环，循环次数为 248 次，可以很容易推导出本例的延时时间 T 的表达式：

T={1+[1+1+2×248+2]×200+2}×1us=100003μs≈0.1s

如果想获得更长的延时时间，可以增加循环的层次，同时通过对每层循环的循环体做适当修正，可以对延时时间做微调，提高延时精度。

编写多重循环程序时必须注意循环嵌套必须层次分明，不允许发生内外层循环交叉。

2.3.3　汇编语言程序设计方法

从前面的内容，可能读者们已经发现，用汇编语言设计程序和在 PC 机上用 C、BASIC 等高级语言设计程序有很多不同，因为用汇编语言设计程序要求设计者对单片机内部结构和数据管理方式要比较熟悉。所以对于很多初次接触单片机的读者来说，使用汇编语言设计程序不是一件很轻松的事情。尤其是对一些复杂的控制过程，如果既要求程序占用空间小，又要求执行时间短，那就更难了。但是如果你掌握了汇编语言程序设计的一般方法和步骤，对于尽快入门和提高还是十分有帮助的。

通常，开发单片机应用系统程序可以按照图 2.8 所示步骤进行：

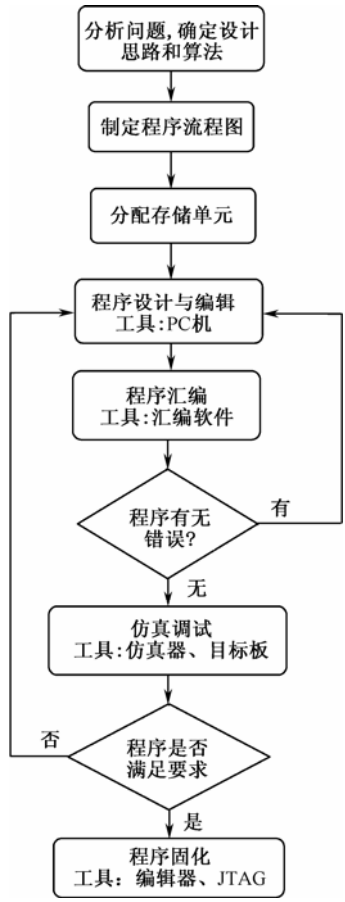


图 2.8　汇编语言程序设计过程

（1）分析问题，确定算法和设计思路。

（2）根据算法和设计思路画出程序流程图。

（3）对单片机资源进行分配，特别是对有限的内部存储单元的要慎重进行。

（4）根据流程图编写源程序。

（5）汇编程序，检查语法错误并更正。

（6）程序仿真调试，找出错误并更正，再调试，直到功能及指标满足要求。

（7）固化或下载程序到单片机。

另外，需要指出的是，随着技术的不断进步，单片机程序调试环境有了很大改变。利用 PC 机及其软件可以模拟各种单片机，形成一个程序编辑、汇编和编译、调试、模拟显示等功

能完善的集成环境，给单片机程序调试带来极大便利。纯软件调试可以完全抛开单片机，甚至有些关于接口电路的调试都可以在这个集成环境中进行。

当今主流单片机其内部一般都有一定数量的 FlashROM，在进行程序调试时，将程序通过 JTAG 或串行口写入单片机内的 FlashROM，无须进行专门的程序固化操作。

2.4 实用程序设计举例

2.4.1 数制转换程序

数制转换在程序设计中常常用到，因为人们习惯十进制数据，而单片机在运算处理时是按照二进制规则进行的。最为常见的是二—十进制数之间的转换。

1. 二进制转换成BCD码十进制

例 2.33 假设在内部 RAM 中 30H 单元内存有一个二进制数，设计一段程序把这个数转换成 BCD 码的十进制数，并将百位数存入 32H 单元，十位和个位数存入 31H 单元。

解：解题思路：一个字节的二进制数，最大不会超过 255，将这个数除以 100，其商为十进制数的百位，再将余数（0～99）除以 10，得到的商即为十进制数的十位，所剩的余数即为十进制的个位数，如下所示：

```
100 | (30H)    商就是 BCD 百位
 10 | 余数     商就是 BCD 十位
    | 余数     余数就是 BCD 个位
```

其转换流程图如图 2.9 所示，程序如下：

```
ORG     0000H
MOV     A，30H
MOV     B，#100
DIV     AB
MOV     32H，A      ；“百位”存入 32H 单元
MOV     A，B
MOV     B，#10
DIV     AB
SWAP    A           ；“十位”存入 A 高 4 位，A3~0=0000
ADD     A，B        ；“个位”存入 A 低 4 位
MOV     31H，A
SJMP    $
END
```

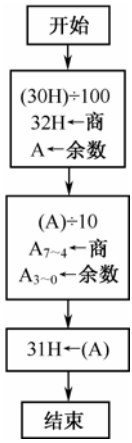


图 2.9 二进制转成 BCD 码十进制

请读者思考：如果要求转换的十进制数不止一个字节，而是两个或者更多，那么该怎样将其转换为二进制数呢？

2. BCD码十进制转换成二进制

例 2.34 假设在内部 RAM 中 40H 单元内存有一个 BCD 码十进制数，设计一段程序把这个数转换成二进制数，并存入 41H 单元。

解：解题思路：一个字节的 BCD 码十进制数，最多能表示 0～99，因此用(40H)÷16，将十进制数的十位和个位分离，商就是十进制数的十位，余数就是十进制数的个位。十位数的权是 10，因此将商乘以 10 变为二进制数，再加上个位数，便是对应的二进制数。

由于这个问题比较简单，所以本例不再画流程图，直接给出程序：

```
ORG     0000H
MOV     A，40H
MOV     B，#16
DIV     AB          ；(40H)÷16，A←商，B←余数，A、B 的高 4 位全 0
MOV     41H，B      ；保存个位
MOV     B，#10
MUL     AB          ；将十位数变成二进制数
ADD     A，41H
MOV     41H，A
SJMP    $
END
```

请读者思考：如果要求转换十进制数比较大，由多个字节组成，该如何编程将其转换为二进制数？

2.4.2 数据处理程序设计

1. 多字节BCD码十进制数相加

例 2.35 假设在内部 RAM 的 30H～37H 单元、38H～3FH 单元分别存放有两个 8 字节 BCD 码十进制数，设计一段程序将这两个数相加，并将结果存放到 2FH～37H 单元中。数据存放格式是小地址存数据的高字节。

解：解题思路：两个 8 字节数据相加，由于最高字节相加时可能产生进位，结果有可能为 9 字节。先清 Cy 位，把(37H)和(3FH)进行带 Cy 相加，再进行十进制调整，结果存入 37H 单元；再把(36H)和(3EH)进行带 Cy 相加，再进行十进制调整……循环至结束，最后把最高字节的进位 Cy 存入 2FH 单元。

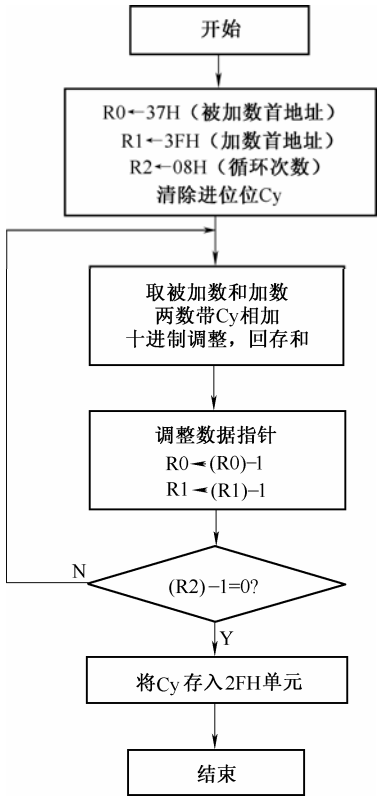


图 2.10 多字节 BCD 码加法流程

流程图如图 2.10 所示。程序如下：

```
        ORG     0000H
        MOV     R2，#08H    ；设定循环次数
        MOV     R0，#37H    ；用 R0 做指针，指向被加数的低位字节
        MOV     R1，#3FH    ；用 R1 做指针，指向加数的低位字节
        CLR     C
LOOP：  MOV     A，  @R0    ；取被加数
        ADDC    A，  @R1
        DA      A
        MOV     @R1，A      ；回存“和”
```

```
        DEC     R0          ；指向被加数的下一个字节
        DEC     R1          ；指向加数的下一个字节
        DJNZ    R2，LOOP
        CLR     A
        RLC     A           ；把最高位的进位存入 2FH 单元
        MOV     2FH，A
        END
```

请读者思考：多字节减法程序该如何编写？特别是多字节 BCD 码十进制数的减法程序。由于 MCS-51 单片机只有单字节乘除指令，多字节的乘除运算必须通过程序来实现，那么这样的程序又该如何编写？

2. 求平均值

例 2.36 在 MCS-51 单片机内部 RAM 的 30H～37H 单元内存有 8 个无符号数，设计一段程序，求这 8 个数的平均数并存入 38H 单元中。

解：解题思路：先将这 8 个数相加，再用和除 8 可得平均数。但是 8 个数相加的和很可能是双字节数，对于像这种除 2^n 的除法运算，可以先把被除数临时存入内部 RAM 单元，然后将和右移 n 位实现除法过程。本例选 R2、R1 临时存放 8 个数的和，然后右移 3 位实现除 8 运算。

流程图如图 2.11 所示。程序如下：

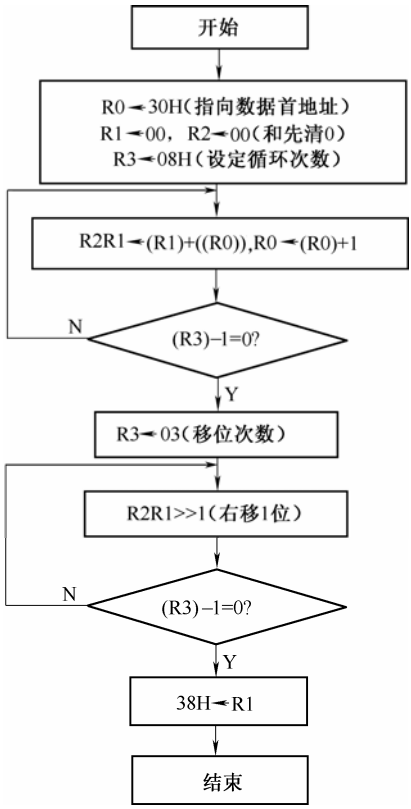


图 2.11 8 字节求平均数流程

```
        ORG     0000H
        MOV     R3，#08         ；设定循环次数，有 n 个字节就循环 n 次
        MOV     R0，#30H
                                ；用 R0 做数据指针，指向数据区首地址
        MOV     R1，#00H        ；和的低 8 位预置为 0
        MOV     R2，#00H        ；和的高 8 位预置为 0
ADD0：  MOV     A，@R0          ；(R2R1)←(R1)+((R0))
        ADD     A，R1
        JNC     ADD1            ；如果(R1)+((R0))＞255，则(R2)←(R2)+1
        INC     R2
ADD1：  MOV     R1，A
        INC     R0              ；调整指针，指向下一个数据
        DJNZ    R3，ADD0        ；循环直到处理完 8 个字节的数据
        MOV     R3，#03         ；设定移位次数，右移 3 位相当于除 8
ROT0：  CLR     C
        MOV     A，R2           ；先把和的高 8 位 R2 右移 1 位
        RRC     A
```

```
        MOV     R2，A
        MOV     A，R1           ；再把和的低 8 位 R1 带进位位右移 1 位
        RRC     A
        MOV     R1，A
        DJNZ    R3，ROT0        ；循环 3 次
        MOV     38H，R1         ；把结果存入 38H
        END
```

请读者思考：前例的除数为 2^n，如果计算平均数的个数为 2^n-1，2^n-2，…，程序该如何修改？

3. 极值查找

例 2.37 在内部 RAM 的 40H～47H 单元内存有 8 个无符号数，设计一段程序找出其中的最大值并存入 48H 单元中。

解：解题思路：先把(A)←(40H)，然后用(41H)和(A)进行比较，如果(41H)＞(A)，则(A)←(41H)，然后和下一字节进行比较，以此类推，最后(A)必定是最大值。

流程如图 2.12 所示。程序如下：

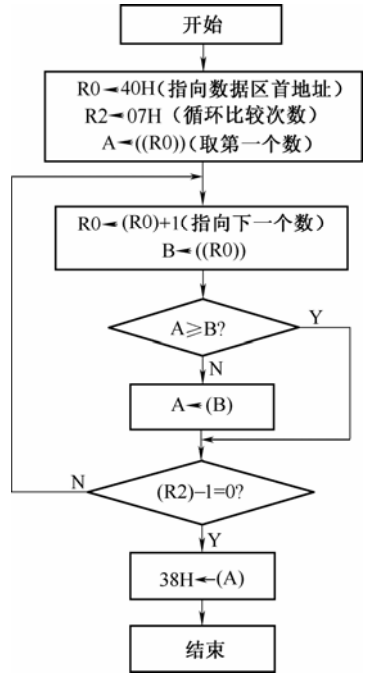


图 2.12 极大值查找程序

```
        ORG     0000H
        MOV     R0，#40H        ；R0 做数据指针，指向
                                  数据区第一个字节
        MOV     R2，#07H        ；R2 做循环计数器
        MOV     A，  @R0        ；先把(A)←(40H)
LOOP：  INC     R0
        MOV     B，@R0          ；取下一字节，临时存入 B 寄存器中
        CJNE    A，B，NEXT
NEXT：  JNC     NEXT1           ；如果(A)＜(B)，则(A)←(B)
        MOV     A，B
NEXT1： DJNZ    R2，LOOP
        MOV     38H，A
        JSMP    $
        END
```

请读者思考：如果要查找的数据串是带符号数，要查找极大值或极小值该如何实现？

4. 关键字查找

关键字查找是在数据区中查找一个特定数据，也称为数据检索。常用的数据检索方法是顺序查找法，即将关键字与数据区中的数据按顺序逐个进行比较，判断是否为所查找的关键字。如果数据是有序排列，也可以利用折半查找方法，这样可以大大提高查找速度。

例 2.38 在 20H 开始的 10 个单元中查找是否存在关键字“0AH”，若存在，则将 1 送入 30H，否则将－1 送到 30H 中。

解：设 R0 为地址指针，初值为 10H。R1 为循环计数器，初值为 10。程序流程图如图 2.13 所示。

程序如下：

```
        ORG     1000H
        MOV     R0，#20H          ；地址指针赋初值
        MOV     R1，#10           ；计数器赋初值
LOOP：  MOV     A，@R0            ；取下一个数据
        CJNE    A，#0AH，NEXT     ；未找到关键字转到 NEXT
        MOV     30H，#01H         ；找到关键字，30H 赋 1
        SJMP    EXIT
NEXT：  INC     R0
        DJNZ    R1，LOOP          ；未查找结束，继续查找
        MOV     30H，#0FFH        ；未找到，30H 赋–1
EXIT：  SJMP    $
        END
```

5. 数据排序

排序是使一组记录按照其关键字的大小，有序地排列起来，排序的方法很多，下面介绍常用的冒泡排序法。

冒泡排序法是在有 N 个数据的数列中依次比较两个相邻的一对数据，如果不符合规定的递增（或递减）顺序，则交换两个数据的位置，第一对（第一个和第二个数据）比较完毕后，接着比较第二对（第二个和第三个数据），直到数列所有的数据依次比较完毕后，第一轮比较结束，这时最大（或最小）的数据降到数列中最后的位置。第一轮排序需要进行（N−1）次比较。同理，第二轮比较需要进行（N−2）次比较，第二轮结束后，次最大（或最小）的数据排在底部往上第二位置上。重复上述过程，直至全部排完，从而实现这组数据由大到小（或由小到大）的顺序排列，数据的排序过程如图 2.14 所示。理论上须要 N-1 轮排序，实际上有

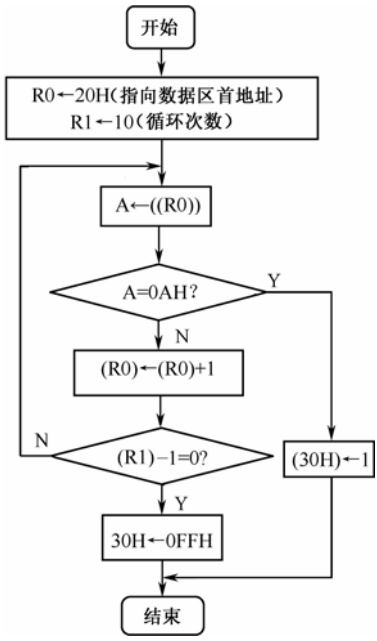


图 2.13　关键字查找程序

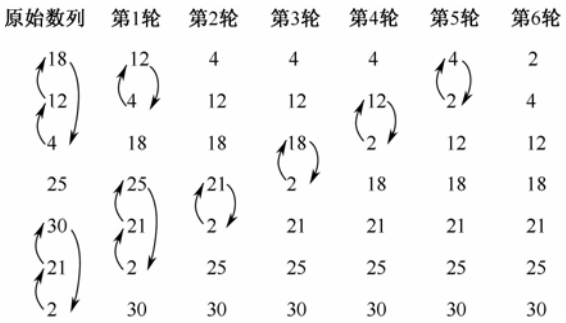


图 2.14　冒泡排序过程

时不到 N−1 轮就已完成排序。如何判定排序是否已经完成了呢？可以通过观察每轮排序中是否有数据交换，如果有交换，则排序未完成；否则就表示排序完成。在程序设计中通常采用设置数据交换标志的方法表示每轮排序是否有数据交换。

例 2.39　将单片机片内 RAM20H～2FH 单元的数据从小到大升序排列。

解：解题思路：采用冒泡法排序，第 1 轮首先找出最大数放到数据块最后，第 2 轮再找次大数排在倒数第 2 位置，直到全部排完为止。采用双重循环结构，设 R7 为比较计数器，初值为 0FH。F0 为每轮排序中数据交换标志，F0=1 表示该轮有数据交换，须要继续循环排序；F0=0 表示该轮无数据交换，排序完毕，可以结束循环。R0 为地址指针，初值为 20H。

流程图如图 2.15 所示。程序如下：

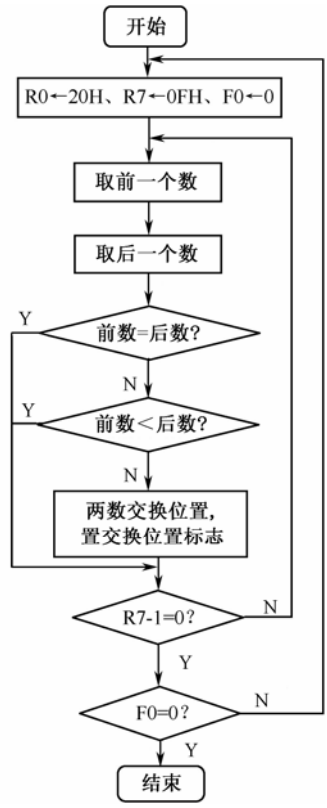


图 2.15　冒泡排序程序

```
            ORG    1000H
SORT：      MOV    R0，#20H        ；数据首地址送 R0
            MOV    R7，#0FH        ；比较次数送 R7
            CLR    F0              ；数据交换标志清 0
LOOP：      MOV    A，@R0          ；取前一个数
            MOV    30H，A
            INC    R0              ；取后一个数
            MOV    31H，@R0
            CLR    C
            CJNE   A，31H，EXCH
            SJMP   NEXT            ；两数相等不交换
EXCH：      JC     NEXT            ；前一个数小于后一个数，则不交换
            MOV    @R0，30H        ；前一个数大于后一个数，则交换
            DEC    R0
            MOV    @R0，31H
            INC    R0
            SETB   F0              ；置交换标志
NEXT：      DJNZ   R7，LOOP        ；本轮未比较完，则继续下一次比较
            JB     F0，SORT        ；本轮有交换标志，继续下一轮排序
            SJMP   $
            END
```

2.4.3　查表程序设计

假如用 MCS-51 单片机编写一段程序实现“200×cos(x)”，读者一定会觉得很困难。是的，但这不是你学艺不精，而是 MCS-51 这种 8 位单片机的计算功能相对较差，如果要用它来实现一些复杂的运算，计算程序编写起来的确相当复杂。对于类似问题，通常可以采用查表的

方法进行。程序是这样设计的：设计者先把对应于不同自变量的输出值计算出来，用 DB 或 DW 伪指令，把这些值有规律地存放在 ROM 中，当要得到某个自变量所对应的函数值时，用 MOVC 指令查表即可。

用查表程序可以非常快捷地完成一些复杂的计算和推理，当然这实际上是设计者预先帮助单片机完成了计算和推理过程，把结果变成一张表格，单片机要做的工作就是查表。

例 2.40 MCS-51 单片机内部 RAM40H 单元内存放的是一个角度参数θ（范围 0°～90°），设计一段程序，计算出 200×sin(θ)，把结果存入 41H 单元中（结果只取整数）。

解：依照查表的思路，程序如下：

```
        ORG   0000H
        MOV   DPTR，#TABLE      ；DPTR 指向表格首地址
        MOV   A，40H
        MOVC  A，@A+DPTR        ；查表，获得相应结果
        MOV   41H，A
TABLE:  DB    0，3，7，10，14，17，21，24，28，31，35，38
        DB    41，45，48……199，199，200，200
        END
```

从 TABLE 开始存放的就是事先由设计者计算的输出结果。

例 2.41 设有一个以 AT89C51 单片机为核心的巡回检测装置，对 8 路信号进行巡回检测，检测信号经处理后送到单片机的 20H、21H，均占 2 个字节，编写程序要求每一路检测信号如果超过最大值则报警，各路信号最大值存放在 ROM 中备查。

解：设信号路数存放在 R2 中，查表得到的某路信号最大值放在 R3 和 R4 中，首先比较检测信号高 8 位（20H）和最大值的高 8 位 R3，如果（20H）>R3，则报警；如果（20H）=R3，比较检测信号低 8 位（21H）和最大值的低 8 位 R4，如果（21H）>R4，则报警。

程序如下：

```
        ORG   1000H
        MOV   DPTR，#TABLE          ；DPTR 指向表格首地址
        MOV   R2，#2
        MOV   A，R2
        ADD   A，R2                 ；(A)←(A)×2，查表地址修正
        PUSH  ACC                   ；A 暂时保存
        MOVC  A，@A+DPTR            ；查表，获得最大值高 8 位
        MOV   R3，A
        POP   ACC
        INC   A                     ；指向下一个字节
        MOVC  A，@A+DPTR            ；查表，获得最大值低 8 位
        MOV   R4，A
        MOV   A，R3
        CJNE  A，20H，NEXT          ；比较高 8 位
        MOV   A，R4
        CJNE  A，21H，NEXT          ；比较低 8 位
NEXT:   JC    ALARM
```

```
        SJMP    EXIT
ALARM：CLR     P1.0                              ；报警
EXIT：  RET
TABLE： DW      1500H，1680H，2010H，1500H    ；8 通道最大值表
        DW      1230H，1780H，2100H，2250H
        END
```

2.4.4 子程序设计

稍微有一点程序设计基础的读者可能都知道，在程序设计过程中，经常有一些程序段会被频繁地使用，比如显示程序、串口发送/接收、延时程序等。通常情况下，我们将这些程序定义成一个子程序，方便调用、阅读和调试，同时也缩短程序长度。

在编写子程序时要注意以下几点：

（1）子程序的第一条指令地址称为子程序的入口地址。该指令前必须要有标号，标号最好以子程序功能命名，这样会一目了然，例如延时程序常以 DELAY 作为标号。

（2）主程序调用子程序是通过 LCALL、ACALL 指令完成，子程序返回主程序是通过执行 RET 指令完成。

（3）子程序内部如果有控制转移指令，最好使用相对转移指令，以便汇编时生成不随子程序的存放地址改变而改变的代码，子程序可以存放在 ROM 空间的任意位置。

（4）在子程序开始部分要注意现场保护，在退出子程序之前要恢复现场。保护现场和恢复现场操作，一般是用 PUSH 和 POP 指令完成，要遵守“先入后出、后入先出”的原则。

在子程序运行时，不可避免地要改变一些寄存器或数据存储器的内容，有时这些内容是主程序所不可缺少的，因此，在子程序调用时，应该先将有关寄存器或存储器的内容保护起来，子程序返回前再恢复原来的内容，这一过程称为现场的保护与恢复。现场保护通常由堆栈来完成，在子程序的开始部分使用压栈指令，将需要保护的内容压入堆栈；在子程序返回前通过出栈指令，将原有的内容弹出堆栈，送到原来的寄存器或存储器单元中，从而实现了现场的保护与恢复。对于 MCS-51 系列单片机，保护和恢复通用寄存器的内容可以通过选择寄存器工作区域来简单地实现。

1. 子程序参数传递

调用子程序时，主程序应把子程序中使用的有关参数送入约定的位置，子程序运行时，可以从约定的位置得到有关的参数，这类由主程序提供给子程序的参数叫做入口参数。同样，在子程序结束前，也应把运算结果送到指定的位置，返回主程序后，主程序可以从指定的位置得到需要的结果，这类由子程序返回主程序的参数叫做出口参数。实现参数传递有多种方法，常用的方法有以下三种：

（1）用工作寄存器或累加器传递参数：这种方法是把入口参数或出口参数存入工作寄存器 Rn 或累加器中。其优点是程序简单，运算速度快。缺点是寄存器数量有限，传递参数的数量较少。

（2）用指针寄存器传递参数：由于数据通常存放在存储器中，因此可以通过使用指针寄存器指示数据的位置来传递数据。这样做的优点是可以传送较多的数据。如果参数在内部 RAM 中，可以使用 R0 或 R1 作为数据指针，如果参数在外部 RAM 中，可以使用 DPTR 作

指针。

（3）用堆栈传递参数：堆栈实际是内部 RAM 的一个区域，当然也可以作为传递参数的工具。使用堆栈进行参数传递时，主程序使用 PUSH 指令把参数压入堆栈中，进入子程序后可以通过堆栈指针用 POP 指令来间接访问堆栈中的参数；用同样的方法，子程序可以把出口参数用 PUSH 压入堆栈中，返回主程序后，可以使用 POP 指令得到这些参数。这种方法的优点是简单易行，并可传递较多的参数。但是要特别注意保护和恢复现场对堆栈的影响。

例 2.42 在外部 RAM 的 00H～07H 单元中依次存放有 8 个无符号数 Xi (i：0～7)，设计一段程序，①计算出 $Yi=Xi^2$，并把 Yi 依次存放到从外部 RAM 10H 开始的 16 个单元中(Yi 占用两个字节，高位在前)；②Zi= Xi÷2，并把 Zi 依次存放到 Xi 所在单元。

解：在本例中，计算 $Yi=Xi^2$ 和 Zi= Xi÷2 这两项工作都是重复劳动，我们可以把它们分别设计成子程序，命名为 SQUARE 和 DIVIDE，程序如下：

```
        ORG     0000H
        MOV     R0，#00H      ；R0 做数据指针，指向 Xi
        MOV     R1，#10H      ；R1 做数据指针，指向 Yi
        MOV     R2，#08H      ；R2 做循环计数器
LOOP：  MOVX    A，@R0        ；取 Xi
        LCALL   DIVIDE        ；调用除 2 子程序
        LCALL   SQUARE        ；调用平方子程序
        MOVX    @R1，B        ；存放 Yi 的高 8 位
        INC     R1
        MOVX    @R1，A        ；存放 Yi 的低 8 位
        INC     R0
        DJNZ    R2，LOOP      ；循环 8 次
        SJMP    $
DIVIDE：PUSH    ACC           ；现场保护，保护累加器
        CLR     C
        RRC     A             ；(A)←(A)÷2，得到 Zi
        MOVX    @R0，A        ；回存 Zi
        POP     ACC           ；恢复现场
        RET                   ；返回
SQUARE：MOV     B，A          ；平方子程序，入口参数为 A
        MUL     AB            ；求平方
        RET                   ；返回，出口参数为 A(低 8 位)和 B(高 8 位)
        END
```

在本例中，在 DEVIDE 子程序的开始部分，用 PUSH 指令把累加器压入堆栈，这就是现场保护，在子程序退出前，用 POP 指令把累加器的内容恢复，这就是恢复现场。为什么在 DEVIDE 子程序里要进行保护和恢复现场的操作呢？请读者注意，程序中，Xi 即(A)先要送给 DEVIDE 子程序进行运算，然后送给 SQUARE 子程序。假如程序从 DEVIDE 退出时，Xi 的值已经被修改了，那么 SQUARE 子程序计算的 Yi 就肯定不正确了。在 DEVIDE 子程序里用 A 参与了运算，而且(A)也被修改了，为了保证传给 SQUARE 的 Xi 不受影响，所以在 DEVIDE 子程序的开始就把 Xi 压入堆栈，保护起来，等退出子程序之前再把它弹出堆栈，恢

复 Xi。那么为什么在 SQUARE 子程序中却没有进行保护和恢复现场呢？因为在 SQUARE 子程序计算出 Yi 之后，不再用 Xi 进行其他的运算和操作，所以即使修改了 Xi 即累加器中的内容也无所谓了。如果我们把 DEVIDE 子程序中的 PUSH 和 POP 语句去除，请读者推算一下 Yi 分别是多少，验证是否正确。

2.4.5 延时程序设计

在设计单片机应用程序时，经常会有延时的问题，比如在例 2.17 中提出了发光二极管每 0.5 秒切换一次显示模式的要求，这就是一个典型的延时问题。MCS-51 单片机实现延时有两种方法：①用单片机内部的定时器/计数器实现；②用软件延时的方法。本节介绍软件延时方法，所谓软件延时，就是通过执行程序延时时间。

1. 单重循环延时程序

例 2.43 假设单片机晶振频率 f_{osc}=12MHz，计算单片机执行下面这段子程序将消耗多长时间。

```
DELAY1:   MOV   R2，#TIME      ；第①条语句，1 个机器周期
LOOP:     NOP                  ；第②条语句，1 个机器周期
          DJNZ  R2，LOOP       ；第③条语句，2 个机器周期
          RET                  ；第④条语句，2 个机器周期
```

解：本例中，由于 f_{osc}=12MHz，所以 1 个机器周期=12×1/(12×10^6)=1μs。程序在执行过程中，第①条语句和第④条语句只执行 1 次，第②、③条语句要执行 TIME 次，由此可以计算出执行这段程序，CPU 要消耗的时间为：

$$T=[1+(1+2)\times TIME+2]\times 1\mu s$$

当 TIME=01H 时，延时时间最短：T=6μs；当 TIME=00H(相当于 256)时，延时时间最长：T=771μs。

2. 多重循环延时程序

在例 2.43 中，如果我们想加长延时时间，最简单的办法就是在 LOOP 开始的循环体内增加 NOP 指令的个数，或增加循环次数 TIME。如果增加的延时很多，达到若干倍，则最有效的办法是采用多重循环。

例 2.44 假设单片机晶振频率 f_{osc}=12MHz，计算单片机执行下面这段子程序将消耗多长时间：

```
DELAY2:   MOV   R3，#TIME1     ；第①条语句，1 个机器周期
LOOP1:    MOV   R2，#TIME2     ；第②条语句，1 个机器周期
LOOP2:    NOP                  ；第③条语句，1 个机器周期
          DJNZ  R2，LOOP2      ；第④条语句，2 个机器周期
          DJNZ  R3，LOOP1      ；第⑤条语句，2 个机器周期
          RET                  ；第⑥条语句，2 个机器周期
```

解：把本例和例 2.43 相对照，实际上就是把例 2.43 中的第②条语句 NOP 换成了本例的由第③④⑤条语句构成的循环体，所以本例实际上由两层循环构成，LOOP1 是外层循环，循环次数为 TIME1 次；LOOP2 是内层循环，循环次数为 TIME2 次。参照例 2.43，可以很容易推导出本例的延时时间 T 的表达式：

$T=\{1+[1+(1+2)\times TIME2+2]\times TIME1+2\}\times 1\mu s$

当 TIME1 和 TIME2 都取 01H 时，延时时间最短，$T=9\mu s$；当 TIME1 和 TIME2 都取 00H 时，延时时间最长，$T=197379\mu s$。

如果想获得更长的延时时间，可以增加循环的层次，同时通过对每层循环的循环体做适当修正，可以对延时时间做微调。

例 2.45　设计一段程序，让 MCS-51 单片机 P1.0 循环输出图 2.16 所示波形（f_{osc}=12MHz）。

图 2.16　P1.0 输出波形图

解：从波形图不难发现，图中各个时间段都是 1ms 的整数倍，我们在设计程序时，可以只编写一个延时 1ms 的子程序，各个不同的延时时间可以通过循环调用该子程序来实现。程序如下：

```
                ORG         0000H
BEGIN:          SETB        P1.0
                MOV         A，#01H
LOOP0:          MOV         R7，A
LOOP1:          LCALL       DELAY1MS
                DJNZ        R7，LOOP1
                CPL         P1.0
                INC         A
                CJNE        A，#07H，LOOP0
                SJMP        BEGIN
DELAY1MS:       MOV         R1，#249      ；延时 1ms 子程序
LOOP3:          NOP
                NOP
                DJNZ        R1，LOOP3
                NOP
                RET
                END
```

本 章 小 结

本章内容是本课程的核心内容之一，是单片机应用的前提和基础。本章内容比较抽象与枯燥，是学习的难点之一，是学习领会单片机基本知识、达到成功应用的“瓶颈”，理论联系实际，边学边练是掌握单片机指令系统的最有效途径。本章要求：了解 MCS-51 单片机的 7 种寻址方式；掌握数据传送与交换、算术运算、逻辑运算、控制转移、位操作 5 类共 111 条指令的功能，特别是每一类指令中比较常用的指令；掌握 MCS-51 单片机汇编语言程序的基本格式及伪指令；了解程序的基本结构（顺序结构、分支结构、循环结构）和程序设计的基本方法；掌握数制转换、算术运算、极值查找等典型程序的设计；掌握子程序的设计方法及其参数传递方法。

习 题 2

一、填空题

2.1 MCS-51 单片机指令系统中，按指令所占用的字节来分有____字节指令、比如_____，和____字节指令、比如_____，以及_____字节指令、比如_____；按执行指令所花费的时间来分有_____周期指令、比如_____；和_____周期指令、比如_____；以及_____周期指令、比如_____；在这里提到的周期是_____周期，它和单片机系统晶振频率之间的关系是_____。

2.2 分析以下各条指令分别是何种寻址方式。

ADD A，B	_____	XCH A，@R1	_____
ORL A，#0FH	_____	XRL 10H， A	_____
MOV C，P1.0	_____	MOVX A，@R0	_____
CJNE A，B，LOOP	_____	DEC 30H	_____
ACALL DELAY	_____	MOVC A，@A+PC	_____

2.3 分析程序运行过程，在每条语句右边的空格中写出在执行完该语句后相应单元的结果。

MOV 40H，#10H	(40H)=_____		
MOV R0， #40H	(R0)= _____		
MOV 10H，#2FH	(10H)=_____		
MOV A，10H	(A)= _____	(10H)=_____	
XCH A，@R0	(A)= _____	(R0)= _____	(40H)=_____
XCHD A，10H	(A)= _____	(10H)=_____	
MOV @R0，A	(A)= _____	(R0)= _____	(40H)=_____
SWAP A	(A)= _____		
MOV R0， A	(A)= _____	(R0)= _____	

2.4 假设已知(SP)=60H，(A)=20H，(20H)=78H，分析下面这段程序运行过程，在每条语句右边的空格中写出在执行完该语句后相应单元的结果。

PUSH ACC	(A)= _____	(SP)= _____	((SP))= _____
PUSH 20H	(20H)=_____	(SP)= _____	((SP))= _____
POP ACC	(A)= _____	(SP)= _____	((SP))= _____
POP 20H	(20H)=_____	(SP)= _____	

2.5 假设已知 (A)=9AH，(R0)=30H，(30H)=7CH。执行指令

ADD A，@R0

后，(A)=_____，Cy=____，OV=____，P=____，AC=_____。

2.6 假设已知 (A)=39H，R2=86H，Cy=1。执行指令

SUBB A，R2

后，(A)=_____，Cy=____，OV=____，P=____，AC=_____。

2.7 假设已知 (A)=25H，(B)=56H，执行指令

MUL AB

后，(A)=_____，(B)= _____，Cy=____，OV=____。

2.8　假设已知 (30H)=17H，(31H)=38H，分析下面这段程序运行过程，在每条语句右边的空格中写出在执行完该语句后相应单元的结果。

```
MOV     A，  30H          (A)=_____
ADD      A，31H           (A)=_____
DA       A                (A)=_____
```

2.9　假设已知 (30H)=17H，(31H)=38H，Cy=1 分析下面这段程序运行过程，在每条语句右边的空格中写出在执行完该语句后相应单元的结果。

```
MOV    R0，#30H          (R0)= _____
MOV    A，  @R0          (A)= _____      (R0)= _____
INC    R0                (R0)= _____
ANL     A，@R0           (A)= _____      (R0)= _____      (31H)= _____
ORL     A，@R0           (A)= _____      (R0)= _____      (31H)= _____
RLC     A                (A)= _____
XRL     A，30H           (A)= _____
RR      A                (A)= _____
INC      R0              (R0)= _____
MOV    @R0，  A          (A)= _____      (R0)= _____      (32H)= _____
```

2.10　分析下面这段程序运行过程，在每条语句右边的空格中写出在执行完该语句后相应位的状态。

```
SETB     P1.0            P1.0= _____
CLR      P1.1            P1.1= _____
MOV      C，P1.0         Cy=  _____
ANL       C，P1.1        Cy=  _____
MOV      P1.2，C         P1.2= _____
CPL      C               Cy=  _____
ORL       C，P1.0        Cy=  _____
MOV      P1.3，C         P1.3= _____
```

P1.2 和 P1.1、P1.0 之间的逻辑关系为：P1.2=_____；P1.3 和 P1.1、P1.0 之间的逻辑关系为：P1.3=_____。

二、程序阅读题

2.11　仔细阅读下面这段程序，看看其中有那些错误：

```
          ORG    0000H
TEMP      BIT    #00H
          MOV    A，  TEMP
SWAP；    ANL    A，  C
          MOV    10H，#CFH
2ST；     MOV    R2，R1
          DJNZ   10H，2ST
          SJMP   SWAP
          END
```

2.12　仔细阅读分析下面这段程序，说明它的功能是什么：

```
        ORG    0000H
        MOV    DPTR，#0100H
        MOV    R0，#20H
        MOV    R1，#10H
LOOP：  MOV    A，@R0
        MOVX @DPTR，A
        INC    R0
        INC    DPTR
        DJNZ   R1，LOOP
        END
```

2.13　仔细阅读分析下面这段程序，说明它的功能是什么：

```
        ORG     0000H
        MOV    A，30H
        PUSH   ACC
        ANL    A，#0FH
        MOV    32H，A
        POP    A
        ANL    A，#0F0H
        SWAP   A
        MOV    31H，A
        END
```

2.14　仔细阅读分析下面这段程序，说明它的功能是什么：

```
        ORG    0000H
        MOV    R0，  #30H
        MOV    R2，  #16
        MOV    40H，#00H
LOOP：  CJNE   @R0，#50，NEXT
NEXT：  JNC    MORE
        INC    40H
MORE：  INC    R0
        DJNZ   R2，LOOP
        SJMP   $
        END
```

三、程序设计题

2.15　设计一段程序实现功能：把片外 RAM2000H 单元开始的 40 个单元的数据传送到片外 RAM0000H 开始的 40 个单元中。

2.16　设计一段程序实现功能：把片内 RAM30H～3FH 单元中的 16 个数的存放顺序颠倒过来。

2.17　设计一段程序实现功能：找出从片内 RAM30H 开始的 16 个单元中最小值所在的单元，并将该单元的内容改成 0FFH。

2.18　编写一段程序实现逻辑运算：P1.7=P1.6(P1.5⊙P1.4)。

2.19　参照图 2.2，设计一个彩灯控制程序，让 8 个 LED 按照你的设想发光，显示的样式尽可能丰富。

2.20　设计一段程序实现功能：统计出从片内 RAM30H 开始的 16 个单元中有多少个数大于 40 小于 80，并把结果存入 40H 单元中。

2.21　在内部 R7 单元中存放的是一个参数 x（范围 0～10），设计一段程序，计算出 $7x^4+6x^3+5x^2+4x+3$，把结果存入 41H、42H 和 43H 单元中（高位在前）。

2.22　设计一个能将十六进制数转换成 ASCII 码的子程序，命名为 HEXASC；用此子程序将(R0)中的 2 位十六进制数转换成 ASCII 码并分别存入 R1(高位)和 R2(低位)。

2.23　R2 中存放有 2 位 BCD 码十进制数，设计一个名为 DISPLAY 的子程序，查出它们的 7 段共阳数码管显示代码并分别存入内部 RAM 的 20H（高位）和 21H（低位）单元中。

2.24　设计一个延时 1 秒的子程序（f_{osc}=6MHz）。

2.25　设计一段程序，让单片机 P1.0 端输出频率为 100Hz 的方波信号（f_{osc}=12MHz）。

2.26　设计一段程序，让单片机 P1.1 端输出频率为 100Hz，占空比为 0.6 的矩形波信号（f_{osc}=12MHz）。

2.27　在内部 RAM30H 单元中存放有一个有符号数，设计一段程序计算出它的绝对值，并把结果存入源地址。

2.28　在片外 RAM2000H～200FH 内存放有 16 个无符号数，编写一段程序，计算出它们的平均值，并把结果存入片外 RAM2010H 单元。

第 3 章　MCS-51 系列单片机中断系统

本章主要内容

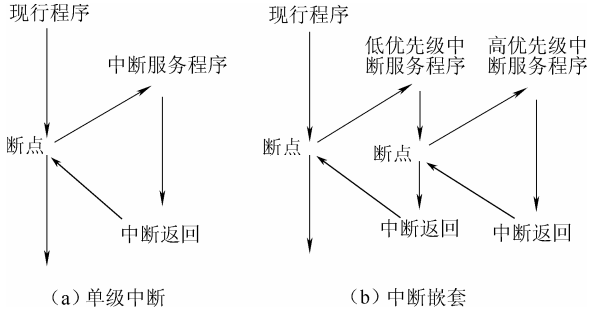
中断及相关的基本概念。MCS-51 系列单片机中断系统。MCS-51 单片机外部中断及应用。

众所周知，CPU 的工作速度愈来愈快，CPU 启动外部设备输入/输出一个字节数据只需要微秒级甚至更短的时间，而低速的外设工作速度一般在毫秒级，若 CPU 和外部设备是串行工作的，则 CPU 就浪费了很多时间去等待外设，其效率大大降低。若没有中断技术，CPU 难于为多个设备服务，对故障的处理能力也极差。为了解决这些问题，增强计算机的能力，在计算机中引入了中断技术，目前所有的计算机都有中断处理的能力。一个功能强大的中断系统，能大大提高单片机处理事件的能力，提高单片机效率，增强单片机的实时性。MCS-51 单片机的中断系统功能较强，51 子系列有 5 个中断矢量：定时器/计数器 0（T0），定时器/计数器 1（T1），外部中断 0（$\overline{\text{INT0}}$），外部中断 1（$\overline{\text{INT1}}$）和串行通信（TI、RI），52 子系列还增加了定时器/计数器 2 中断矢量。MCS-51 单片机的中断分为两个优先级，可实现两级中断嵌套。用户可以很方便地通过软件实现对中断的控制。

3.1　中断系统概述

3.1.1　中断系统的概念

中断是 CPU 在执行现行程序的过程中，发生随机事件和特殊请求时，使 CPU 暂停现行程序的执行，而转去执行对随机事件或特殊请求的处理程序，待处理完毕后，再返回被中止的程序继续执行的过程。实现中断的硬件逻辑和实现中断功能的指令统称为中断系统。引起中断的事件称为中断源；实现中断功能的处理程序称为中断服务程序。中断的响应过程如图 3.1 所示，图（a）为单级中断，图（b）为两级中断嵌套。



（a）单级中断　　（b）中断嵌套

图 3.1　CPU 响应中断过程的示意图

对于中断系统来说，引起中断的事件称为中断源；由中断源向 CPU 所发出的请求中断的信号称为中断请求信号；CPU 暂停现行程序执行的位置称为中断断点；中断断点处的 CPU

各寄存器的状态称为中断现场；由中断服务程序返回到原来程序的过程称为中断返回；CPU接受中断请求而暂停现行程序，转去为中断源服务称为中断响应。

1．中断源

通常，单片机系统的中断源有下列几种：

（1）设备中断。单片机应用系统常有外部设备或外围设备，当这些设备需要 CPU 服务时可以向 CPU 发出中断请求。如键盘、打印机、A/D 转换器等。

（2）实时时钟或计数信号。如定时时间或计数次数一到，则向 CPU 发出中断请求，要求 CPU 予以处理。如定时控制或定时数据采集系统等。

（3）故障源。当单片机系统出现故障时，向 CPU 发出中断请求，要求 CPU 加以处理。如系统停电，即当单片机系统直流电源的电压下降到一定程度时，通过电压监视电路获得此信号，向 CPU 发出中断信号，单片机在中断服务程序中及时进行重要数据保存或切换 RAM 的后备电池，以便保护 RAM 中的信息不丢失。

（4）为调试程序而设置的中断源。为了便于控制程序的调试，及时检查中间结果，可以在程序中设置一些断点或单步执行等。

对于中断源来讲，不仅要求能发出中断请求信号，还要求中断请求信号维持一定时间，直到 CPU 响应中断请求后才能撤除这个中断请求信号。但是，中断请求信号也不能保持太长时间，否则会引起重复中断。一般地讲，每一个中断源的接口电路中有一个中断请求触发器，在多个中断源情况下，为了增加控制灵活性，在每个中断源接口电路中还设置了一个中断屏蔽触发器，由它控制该中断源的中断请求信号能否送到 CPU。

2．中断优先级

随着中断技术的发展，在中断系统中设有多个中断源。当有两个以上的中断源同时请求中断时，就存在 CPU 该如何处理的问题。要解决这个问题就提出了中断优先级，将不同中断源划分成不同的优先级别，CPU 响应优先级别高的中断请求。不同的中断系统有不同的处理方法，主要有硬件中断优先级排队电路、优先权比较电路、按优先顺序查询、软件随机设定等方法。

3．中断识别方式

单片机中断源的数目反映了该单片机处理中断的能力。由于单片机一般都设有多个中断源，中断系统必须具备正确识别中断的能力，才能可靠为其服务。中断的识别方式一般有两种：查询中断和矢量中断。

查询中断识别方式是通过软件逐个查询各中断源的中断请求标志，其查询顺序反映了中断源的优先顺序。先查询的优先级高，后查询的优先级低。通过查询找出申请中断的中断源，然后转向相应的中断服务程序去执行。这种方式的优点在于中断源的优先级别完全取决于程序，用户可以方便地进行修改或设定。但这种方式的缺点也是显然的，由于软件查询需要占用一定时间，而且当有任何一个中断源有中断请求时，都必须从优先级最高的中断开始查询，逐步向优先级低的中断查询，影响 CPU 响应中断的效率和速度，优先级低的中断源的中断请求得到响应的几率将受到影响。

矢量中断识别方式则以硬件为基础，单片机为每个中断源直接提供相对应的中断服务程

序入口地址。中断请求通过优先级排队电路，一旦某中断请求通过了优先级排队电路，说明它是当前最高级别的中断请求，如果此时 CPU 满足中断响应条件，就立即转向对应的矢量地址去执行。因此，矢量中断识别方式的响应速度快，提高了 CPU 的响应效率。

4．中断响应

当 CPU 收到中断请求申请后，如果没有更高级别的中断请求，且满足中断响应的条件，则在执行完当前指令（与中断系统无关的指令）后响应这一请求。响应过程应包括：保护断点、保护现场、执行相应的中断服务程序。

当 CPU 响应某一中断的请求，在进行中断处理时，若有优先权级别更高的中断源发出中断请求，CPU 要将中断正在进行的中断服务程序，保留这个程序的断点和现场，而响应更高优先级的中断请求，在更高优先级的中断处理完以后，再继续执行被中断的中断服务程序，即形成中断嵌套，如图 3.1（b）所示。而当发出新的中断请求的中断源的优先级与正在处理的中断源同级或更低时，则 CPU 就不响应这个中断请求，直至正在处理的中断服务程序执行完以后才去处理新的中断申请。当中断服务程序执行完毕后返回被中断的程序继续执行。

当中断服务完成后，为了能返回到原来操作继续执行，需要恢复现场和恢复断点（执行 RETI 指令）。由于中断发生是随机的，因此，在中断响应时，保护断点是计算机自动完成的。但现场的保护与恢复需要用户通过程序去完成。

3.1.2　中断的作用

中断技术极大地增强计算机处理随机事务的能力，中断主要有以下功能。

1．实现CPU与外部设备的速度配合

在计算机应用系统中，往往存在速度迥异的各种外部设备，而外部设备的工作速度比起 CPU 来讲是十分缓慢的。中断系统可以实现 CPU 和外设并行工作，只有当某外设需要 CPU 服务时才提出服务请求（中断请求），一旦 CPU 响应完该中断请求后又去完成其他工作。这不仅协调了快速 CPU 和慢速外设之间的速度匹配矛盾，而且大大提高计算机系统的工作效率，使得 CPU 能处理多个外设的服务需求，实现分时操作成为可能。

2．实现实时处理

实时处理对于工业控制是十分重要，包括实时数据采集、实时数据处理和运算、实时控制等。工业控制是单片机的一个主要用途，利用其中断系统可以对生产过程的随机信息及时采集和处理，实现实时控制。

3．实现故障处理

单片机应用系统由于受到外界的干扰，以及单片机系统自身的硬件或软件设计等因素，在实际运行中会出现硬件故障、运算错误、程序故障、电源故障等，有了中断技术，单片机就能及时发现故障并自动处理，提高单片机系统的故障处理能力。

3.2　MCS-51 系列单片机中断源与中断请求

MCS-51 系列单片机中不同型号芯片的中断源数量是不同的，最基本的 8051 单片机有 5

个中断源，分别是$\overline{\text{INT0}}$、$\overline{\text{INT1}}$、T0、T1 和串行口。中断源分为两个中断优先权级别，可以实现两级中断服务程序嵌套。每一个中断源可以编程为高优先权级别或低优先权级别中断，允许或禁止向 CPU 请求中断。MCS-51 系列单片机基本的中断系统结构如图 3.2 所示。

由图 3.2 可知，所有的中断源都要产生相应的中断请求标志，这些标志分别放在特殊功能寄存器 TCON 和 SCON 的相关位。每一个中断源的请求信号需经过中断允许 IE 和中断优先权选择 IP 的控制才能够得到单片机的响应。

由图 3.2 我们可以看出，MCS-51 系列单片机有 5 个中断源，它们是：外部中断$\overline{\text{INT0}}$（P3.2）、$\overline{\text{INT1}}$(P3.3)；定时器/计数器 T0、T1 溢出中断；串行口的发送（TXD）和接收（RXD）中断源（只占 1 个中断源）。

外部中断的中断请求标志位和 T0、T1 的溢出中断请求标志位锁存在定时器/计数器控制寄存器 TCON 中，而串行口对应的中断请求位锁存在串行口控制寄存器 SCON 中。

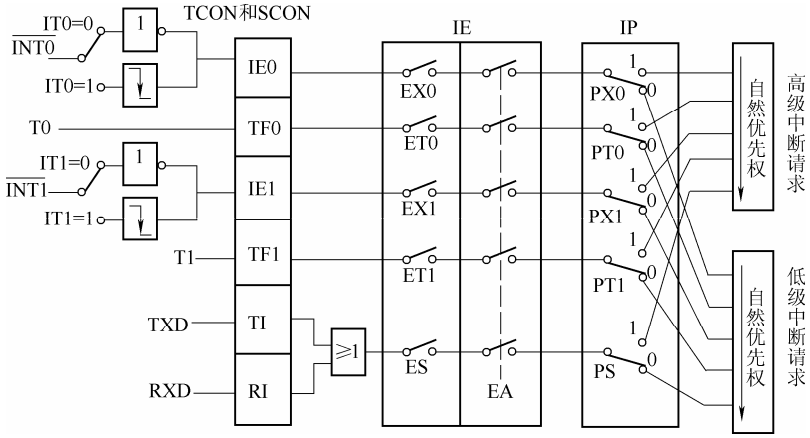


图 3.2　MCS-51 系列单片机 8051 的中断系统内部结构图

3.2.1　定时器/计数器控制寄存器TCON

TCON 为定时器/计数器控制寄存器，其字节映像地址为 88H，可进行位寻址，它除了控制定时器/计数器 T0、T1 的溢出中断外，还控制着两个外部中断源的触发方式和锁存两个外部中断源的中断请求标志。其格式如下：

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TCON 寄存器各位的含义如下：

IT0：外部中断$\overline{\text{INT0}}$的中断触发方式选择位。当 IT0 位清为 0 时，外部中断$\overline{\text{INT0}}$为电平触发方式。在这种触发方式中，CPU 在每一个机器周期的 S5P2 采样$\overline{\text{INT0}}$（P3.2）引脚的输入电平，当采样到低电平时，置$\overline{\text{INT0}}$的中断请求标志位 IE0 位为 1，采样到高电平清 IE0 位为 0。我们在采用电平触发方式时，外部中断源（输入到$\overline{\text{INT0}}$，即 P3.2 引脚）上的必须保持低电平有效，直到该中断被 CPU 响应，同时在该中断服务程序执行结束之前，外部中断源的有效信号必须被清除，否则将产生另一次中断。为了保证 CPU 能正确采样电平状态，要求外部中断源$\overline{\text{INT0}}$有效的低电平信号至少要维持一个机器周期以上。

当 IT0 位置为 1 时，外部中断$\overline{\text{INT0}}$为边沿触发方式。在这种触发方式中，CPU 在每个机器周期的 S5P2 采样$\overline{\text{INT0}}$（P3.2）引脚上的输入电平。如果在相继的两个机器周期，一个

周期采样到$\overline{\text{INT0}}$为高电平，而接着的下一个周期采样到低电平，则置$\overline{\text{INT0}}$的中断请求标志位 IE0 为 1，即当 IE0 位为 1 时，表示外部中断$\overline{\text{INT0}}$正在向 CPU 请求中断，直到该中断被 CPU 响应时，才由硬件自动将 IE0 位清为 0。因为 CPU 在每一个机器周期采样一次外部中断源输入引脚的电平状态，因此采用边沿触发方式时，外部中断源输入的高电平信号和低电平信号时间必须保持在一个机器周期以上，才能保证 CPU 检测到此信号由高到低的负跳变。

IE0：外部中断$\overline{\text{INT0}}$的中断请求标志位。当 IE0 位为 0 时，表示外部中断源$\overline{\text{INT0}}$没有向 CPU 请求中断；当 IE0 位为 1 时，表示外部中断$\overline{\text{INT0}}$正在向 CPU 请求中断，且当 CPU 响应该中断时由硬件自动对 IE0 进行清 0。

IT1：外部中断$\overline{\text{INT1}}$的中断触发方式选择位。功能与 IT0 相同。

IE1：外部中断$\overline{\text{INT1}}$的中断请求标志位。功能与 IE0 相同。

TF0：定时器/计数器 T0 的溢出标志，即中断请求标志。在定时器/计数器 T0 被允许计数后，从初值开始加 1 计数，当产生计数溢出时由硬件自动将 TF0 位置为 1，通过 TF0 位向 CPU 申请中断，一直保持到 CPU 响应该中断后才由硬件自动将 TF0 位清为 0。当 TF0 位为 0 时，表示 T0 未计数或计数未产生溢出。当 T0 工作在不允许中断时，TF0 标志可供程序查询。

TF1：定时器/计数器 T1 的溢出标志，即中断请求标志。功能与 TF0 相同。

TR0 和 TR1 位分别是 T0 和 T1 的运行控制位，其具体功能和作用将在 4.2.2 中介绍。

3.2.2 串行口控制寄存器SCON

SCON 为串行口控制寄存器，其字节映像地址为 98H，也可以进行位寻址。串行口的接收和发送数据中断请求标志位（RI、TI）被锁存在串行口控制寄存器 SCON 中，其格式如下：

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SCON 寄存器各位的含义如下：

RI：串行口接收中断请求标志位。当串行口以一定方式接收数据时，每接收完一帧数据，由硬件自动将 RI 位置为 1。而 RI 位的清 0 必须由用户用指令来完成。

TI：串行口发送中断请求标志位。当串行口以一定方式发送数据时，每发送完一帧数据，由硬件自动将 TI 位置为 1。而 TI 位的清 0 也必须由用户用指令来完成。

注意：在中断系统中，将串行口的接收中断 RI 和发送中断 TI 经逻辑或运算后作为内部的一个中断源。当 CPU 响应串行口的中断请求时，CPU 并不清楚是接收中断请求还是发送中断请求，所以用户在编写串行口中断服务程序时，在程序中必须识别是 RI 还是 TI 产生的中断请求，从而执行相应的中断服务程序。

SCON 其他位的功能和作用与串行通信有关，将在第 7 章中介绍。

在上述的特殊功能寄存器中的所有中断请求标志位，都可以由软件加以控制，即用软件置位或清 0。当某位置位时，就相当于该位对应的中断源向 CPU 发出中断请求，如果清 0 就撤销中断请求。

3.3 MCS-51 系列单片机中断控制

3.3.1 中断允许控制

在计算机中断系统中有两种不同类型的中断：一类为非屏蔽中断，另一类为可屏蔽中断。

对于非屏蔽中断，用户不能用软件方法加以禁止，一旦有中断请求，CPU 就必须予以响应。而对于可屏蔽中断，用户则可以通过软件方法来控制它们是否允许 CPU 去响应。允许 CPU 响应某一个中断请求称为中断开放（或中断允许），不允许 CPU 响应某一个中断请求称为中断屏蔽（或中断禁止）。

MCS-51 系列单片机的 5 个中断源都是可屏蔽中断，没有不可屏蔽的中断。由图 3.2 可知，CPU 对中断源的中断开放或中断屏蔽的控制是通过中断允许控制寄存器 IE 来实现的。IE 的字节映像地址为 0A8H，既可以按字节寻址，也可以按位寻址。当单片机复位时，IE 被清为 0。

通过对 IE 的各位的置 1 或清 0 操作，实现开放或屏蔽某个中断，也可以通过对 EA 位清 0 来屏蔽所有的中断源。IE 的格式如下：

EA	—	—	ES	ET1	EX1	ET0	EX0

IE 寄存器各位的含义为：

EA：总中断允许控制位。当 EA 位为 0 时（CLR EA），屏蔽所有的中断；当 EA 位为 1 时（SETB EA），开放所有的中断。

ES：串行口中断允许控制位。当 ES 位为 0 时（CLR ES），屏蔽串行口中断；当 ES 位为 1（SETB ES）且 EA 位也为 1 时开放串行口中断。

ET1：定时器/计数器 T1 的中断允许控制位。当 ET1 位为 0 时（CLR ET1），屏蔽 T1 的溢出中断；当 ET1 位为 1（SETB ET1）且 EA 位也为 1 时开放 T1 的溢出中断。

EX1：$\overline{\text{INT1}}$ 的中断允许控制位。当 EX1 位为 0 时（CLR EX1），屏蔽 $\overline{\text{INT1}}$；当 EX1 位为 1（SETB EX1）且 EA 位也为 1 时开放 $\overline{\text{INT1}}$。

ET0：定时器/计数器 T0 的中断允许控制位。功能与 ET1 相同。

EX0：$\overline{\text{INT0}}$ 的中断允许控制位。功能与 EX1 相同。

例如，要开放 $\overline{\text{INT1}}$ 和 T1 的溢出中断，屏蔽其他中断，则对应的中断允许控制字为：10001100B，即 8CH。这样我们只要将这个结果送入 IE 中，中断系统就按所设置的结果来管理这些中断源。形成这个控制结果的方法可以对 IE 按字节操作，也可以按位操作。

按字节操作形式	按位操作形式
	SETB EX1
MOV IE，#8CH	SETB ET1
	SETB EA

请读者思考：如果要开放外部中断 0 和串行口的中断，而屏蔽其他中断的控制字是什么？如何来实现这个控制结果呢？

3.3.2 中断优先权管理

在中断系统中，要使某一个中断被优先响应的话，就要依靠中断优先权控制。MCS-51 系列单片机对所有中断设置了两个优先权，每一个中断请求源都可以编程设置为高优先权中断或低优先权中断，从而实现二级中断嵌套。为了实现对中断优先权的管理，在 MCS-51 内部提供了一个中断优先级寄存器 IP，其字节地址为 0B8H，既可以按字节形式访问，又可以按位的形式访问。其格式如下：

—	—	—	PS	PT1	PX1	PT0	PX0

IP 寄存器各位的含义如下：

PX0、PT0、PX1、PT1 和 PS 分别为 $\overline{\text{INT0}}$、T0、$\overline{\text{INT1}}$、T1 和串口中断优先级控制位。当相应的位为 0 时，所对应的中断源定义为低优先级，相反则定义为高优先级。

例如，要将 T0 定义为高优先级，使 CPU 优先响应其中断，其他中断均定义为低优先级，对应的优先级控制字为：00000010B，即 02H。只要将这个控制字送入 IP 中，CPU 就优先响应 T0 产生的溢出中断，并将其他中断按低优先级中断处理。具体操作形式如下：

按字节操作形式	按位操作形式
MOV IP，#02H	SETB PT0

在同一个优先级中，各中断源的优先级别由一个内部硬件查询序列来决定，所以在同级中断中，按硬件查询序列也确定了一个自然优先级，其从高到低的优先级排列如下：

$\overline{\text{INT0}}$→T0→$\overline{\text{INT1}}$→T1→串口（RI、TI）

按中断优先权设置后，响应中断的基本原则是：

（1）若多个中断请求同时有效，CPU 优先响应优先权最高的中断请求。

（2）同级的中断或更低级的中断不能中断 CPU 正在响应的中断过程。

（3）低优先权的中断响应过程可以被高优先权的中断请求所中断，CPU 会暂时中止当前低优先权的中断过程，而优先响应高优先权中断。等到高优先权中断响应结束后再继续响应原低优先权的中断过程，形成中断的嵌套。

为了实现上述功能和基本原则，在 MCS-51 系列单片机中断系统的内部设置了两个不可寻址的优先级触发器，一个是指出 CPU 是否正在响应高优先权中断的高优先级触发器，另一个是指出 CPU 是否正在响应低优先权中断的低优先级触发器。当高优先级触发器状态为 1 时，屏蔽所有的中断请求；当低优先级触发器状态为 1 时，屏蔽所有同级的中断请求而允许高优先权中断的中断请求。

MCS-51 系列单片机复位后，特殊功能寄存器 IE、IP 的内容均为 0，由用户的初始化程序对 IE、IP 进行初始化，开放或屏蔽某些中断并设置它们的优先权。

3.4 中断响应

单片机一旦工作，并由用户对各中断源进行使能和优先权初始化编程后，MCS-51 系列单片机的 CPU 在每个机器周期顺序检查每一个中断源。那么，什么情况 CPU 可以及时响应某一个中断请求呢？若 CPU 响应某一个中断请求，它又是如何工作的呢？

3.4.1 中断响应条件

单片机的 CPU 在每个机器周期的最后一个状态周期采样并按优先权设置的结果处理所有被开放中断源的中断请求。一个中断源的请求要得到响应，必须满足一定的条件。

（1）CPU 正在处理相同的或更高优先权的中断请求；这种情况下只有当前中断响应结束后才可能响应另一个中断请求。

（2）现行的机器周期不是当前所执行指令的最后一个机器周期；此时只有在当前指令执行结束周期的下一个机器周期才可能响应中断请求。

（3）正在执行的指令是中断返回指令（RETI）或者是对 IE、IP 的写操作指令。在这种情况下，只有在这些指令执行结束并至少再执行一条其他指令后才可能响应中断请求。

如果上述条件中有一个存在，CPU 将自动丢弃对中断查询的结果；若一个条件也不存在，

则将在紧接着的下一个机器周期执行中断查询的结果，响应相应的中断请求。

3.4.2 中断响应过程

如果某一个中断被开放，且中断请求符合响应条件，CPU 会及时响应该中断请求，并按下列过程进行处理：

（1）置相应的优先级触发器状态为 1，指明了 CPU 正在响应的中断优先权的级别，并通过它屏蔽所有同级或更低级的中断请求，允许更高级的中断请求。

（2）执行一条硬件子程序调用，清除相应的中断请求标志位（RI、TI 和电平触发的外部中断除外）。

（3）保护断点。即将所中断程序的断点程序位置（PC 的值）压入堆栈保存起来。

（4）获取将被响应的中断源的中断服务程序入口地址，并将其送入程序计数器 PC 来调用该中断服务程序。MCS-51 采用的是固定中断入口地址的方法，即中断入口地址和中断源的编号是一一对应且固定不变，各中断源的中断服务程序入口地址的分配情况如表 1.4 所示。

5 个中断源的中断服务程序入口地址之间只有 8 个字节的空间，一般情况下，8 个字节不足以容纳一个中断服务子程序，因此总是在中断入口地址处放一条无条件转移指令，以转向另外安排的中断服务子程序的入口，以便有足够的空间来安排中断服务子程序。

（5）执行相应的中断服务程序。中断服务程序是用户根据实际应用的需要而编写的应用程序，以中断返回指令（RETI）为结束标志，当 CPU 执行完中断服务程序中的中断返回指令后，清相应的优先级触发器为 0 ，然后恢复断点。即将保存在堆栈中的程序计数器 PC 的值再弹给 PC，使 CPU 再继续执行原来被中断的程序。

中断响应过程的执行都是由 CPU 的中断系统来完成的，对于用户来讲，最重要的是根据实际应用编写中断服务程序。

3.4.3 中断响应的时间

在大多数应用中，我们并不关心中断响应的时间，不过在特殊场合下是非常必要的。在 MCS-51 系列单片机中的外部中断请求信号在每一个机器周期的 S5P2 被采样并锁存到相应的中断请求标志中，这个状态等到下一个机器周期才被查询。如果中断被开放，并符合响应条件，CPU 接着执行一个硬件子程序调用指令以转到相应的中断服务程序入口，该调用指令需要两个机器周期，所以从外部产生中断请求到 CPU 开始执行中断服务程序的第 1 条指令之间，最少需要 3 个完整的机器周期。如果中断请求被阻止，则需要更长的时间。如果已经在处理同级或更高级中断，额外的等待取决于中断服务程序的处理过程。如果正处理的指令没有执行到最后的机器周期，即使是需要时间最长的乘法或除法指令，所需的额外等待时间不会超过 3 个机器周期；如果是 CPU 正在执行中断返回指令 RETI 或对 IE、IP 的写操作指令，加上另外一条指令的执行时间，额外的等待时间不会多于 5 个机器周期。所以在单一中断系统中，外部中断响应时间总是在 3 ～ 8 个机器周期。在实时性要求特别高的应用场合，要充分估计中断响应时间。

3.5 中断请求的撤除

CPU 响应中断请求，转向执行中断服务程序，在执行中断返回指令之前，中断请求信号

必须撤除，否则将引起再一次中断，使系统产生误动作。中断请求撤除的方式有以下几种。

（1）单片机内部硬件自动清除中断标志。对于定时器/计数器 T0、T1 和采用边沿触发方式的外部中断请求，在 CPU 响应中断后，IE0、IE1、TF0 和 TF1 标志由内部硬件自动清除，即硬件置位，硬件清除。

（2）软件清除相应标志。对于串行接收/发送中断请求，RI 或 TI 标志在 CPU 响应中断时没有进行硬件清除，必须用软件清除对应中断标志，即硬件置位，软件清除。

（3）外加硬件清除。对于边沿触发方式的外部中断，中断标志的撤销是自动的，但对于电平触发的外部中断则需要外加硬件电路来撤除。由于电平是持续性的信号，而 CPU 工作速度很快，当 CPU 完成中断响应时，有效的中断请求信号可能还存在，会再次产生中断，引起误动作。如图 3.3 所示的电路，当外部中断请求信号使得 D 触发器 Q 端输出为低电平，产生持续低电平中断请求信号，为了保证在中断响应后把中断请求信号从低电平强制改为高电压（撤出中断请求信号）。在中断服务程序的最前面安排指令，在单片机的 P1.0 引脚产生一个负脉冲，强制 D 触发器的 Q 端置 1，从而清除了中断请求。所需负脉冲可用以下两条指令获得：

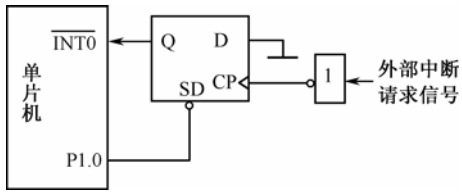


图 3.3　电平方式外部中断请求的撤除电路

```
ANL   P1，#0FEH      ；用与运算使 P1.0 置 0，而不改变其他位的状态
ORL   P1，#01H       ；用或运算使 P1.0 置 1，而不改变其他位的状态
```

还可以通过位操作指令完成，如下所示：

```
CLR   P1.0           ；P1.0 为低电平
SETB P1.0            ；P1.0 为高电平
```

3.6　中断应用举例

3.6.1　中断程序设计基础

中断程序一般包含两个部分，在主程序中对中断初始化，在中断服务程序中实现中断操作任务。在主程序的任何位置都可以进行中断初始化，只有中断初始化程序执行后，中断请求才能得到响应。为了规范，一般在主程序开头安排中断初始化程序，中断服务程序可以放在程序空间的任何位置，但要在相应中断源服务程序入口处放置相应的转移指令。

1．中断初始化

中断初始化包括系统初始状态的设置、开放中断和中断服务程序的前期初始化等。

（1）初始化状态设置。中断初始化状态设置主要是对系统特殊功能寄存器进行设置来保证中断系统的工作状态，如外部中断的触发方式选择，中断源的优先级设置等。如果单片机系统复位状态可以满足应用需要，也可以不进行初始化状态设置。

（2）开放中断。开放中断是通过特殊功能寄存器 IE 来设置的。包含两个部分，一是对每个中断源进行开放设置，若允许某中断源则开放相应的控制位，对于单片机应用系统中没有使用的中断源，一般选择将其屏蔽。二是对单片机进行中断开放设置，即将 EA（IE.7）设置为 1，只有 EA=1，单片机才可能响应中断请求。开放中断是中断初始化中不可缺少的工作。

（3）中断服务程序的前期初始化。有些中断服务程序，它的有关工作需要在主程序中完成。如对定时器/计数器的初始化、计数初值装载以及启动运行等。

2．中断服务程序

中断请求一经得到单片机的响应后，自动保护断点，进行硬件查询，将相应中断源的入口地址（中断矢量）装入 PC，引导中断服务程序。

（1）中断转移。单片机的中断服务程序的入口地址是固定的，一般在入口地址处放置一条无条件转移指令（根据中断服务程序的存放地址，可以使用 AJMP 或 LJMP 指令，LJMP 指令更具一般性），将程序转移至中断服务程序入口处，这样，中断服务程序的存放地址就可以任意安排了。

（2）中断服务。中断服务是中断服务程序的核心，用户要根据自己的应用状况和特点编写相应的程序。但考虑中断服务程序的运行特点，提高系统运行的可靠性，在中断服务程序中主要进行主程序资源保护、中断清除和关中断等操作。

主程序资源保护即保护现场。由于中断产生是随机的，在中断产生时，主程序中的一些数据、寄存器工作组、程序运行标志等信息不能在中断服务程序中被破坏，否则当返回主程序后，主程序不能正确运行。一般为了可靠，凡是在中断服务程序中用到的资源都将其保护起来。保护现场一般使用堆栈指令和选择不同的工作寄存器组来实现。

中断清除是针对单片机不能自动清除的中断源而言的。对于串行通信中断请求 RI 和 TI，需要用指令清除（CLR RI 和 CLR TI），对于电平触发的外部中断，需要通过硬件电路和相应的指令来清除。对于 52 子系列单片机，定时器/计数器 2 的中断请求标志也需要用指令清除。

对于只需要中断一次的中断源，在中断服务程序中必须将其关闭，否则可能会引起再次中断。

（3）中断返回。中断服务程序末尾必须使用中断返回指令 RETI，在 RETI 指令之前一般要根据保护现场的情况恢复现场。RETI 指令将断点处的程序地址自动装入 PC，实现返回被中断的程序继续运行。

3.6.2　外部中断应用举例

如图 3.4 所示，P1 口输出控制 8 只发光二极管，实现 8 位二进制计数器，对 $\overline{\text{INT0}}$ 上出现的脉冲数进行计数。

在图 3.4 中，由与非门构成去抖动电路，K 开关每动作一次，就输出一个脉冲去单片机的 $\overline{\text{INT0}}$ 引脚上，让 $\overline{\text{INT0}}$ 工作于边沿触发方式，每按一次开关 K，就产生一次中断，在中断服务程序中计一次数，并把计数结果从 P1 口输出，显示在发光二极管上。

程序如下：

```
        ORG    0000H
START:  AJMP   MAIN        ；上电转主程序
        ORG    0003H
```

```
        AJMP    EXINT0      ；转 INT0 服务程序
MAIN：  SETB    IT0         ； INT0 为边沿触发方式
        SETB    EX0         ；开放 INT0 中断
        SETB    EA          ；开放 CPU 中断
        CLR     A           ；计数器清 0，并清除显示
        MOV     P1，A
HERE：  SJMP    HERE        ；踏步等待中断
EXINT0：INC     A           ；计数器加 1
        MOV     P1，A       ；输出显示
        RETI                ；中断返回
        END
```

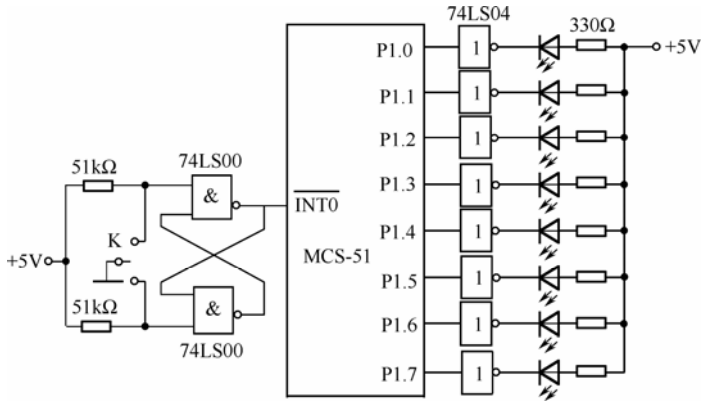


图 3.4　外部中断应用例图

在上例中，由于主程序和中断服务程序都很简单，所以在中断服务程序中没有安排现场保护和恢复，如果在中断服务程序中使用的寄存器和存储单元与主程序有冲突时，必须进行现场保护和恢复。

3.6.3　外部中断的扩展

在 MCS-51 单片机中只有两个外部中断请求输入端 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$。当某个系统需要多个外部中断时，可以通过 OC 门结合软件来扩展，如图 3.5 所示。图中利用 $\overline{\text{INT0}}$ 扩展 4 个外部中断。4 个 OC 门反相器与 1 个上拉电阻构成了“线与”电路，当 4 个外部中断源中有一个或几个出现高电平，则反相器输出为 0，引起 $\overline{\text{INT0}}$ 低电平触发中断。当满足外部中断请求条件时，CPU 响应中断，转入 0003H 单元执行中断服务程序。由于 4 个中断源都通过相同的引脚向 CPU 发出中断请求，因此依靠硬件是无法分清具体是哪一个中断源发出的请求，必须在中断服务程序中，由软件从 P1 口读入各中断源的状态，查询外中断哪一位是高电平，然后进行入该中断的处理程序。查询的顺序就是外部扩展中断源的中断优先级顺序。在这里利用 OC 门构成“线与”逻辑不仅在功能上完全取代了与门的功能，并且比使用与门更节省硬件和方便线路设计。

需要注意的是，外部中断源如此扩展之后，要求所有的扩展中断源的中断请求信号都是电平信号，便于中断响应时查询，当查询结束后，必须对相应请求信号进行撤除，撤除方法如前所述。

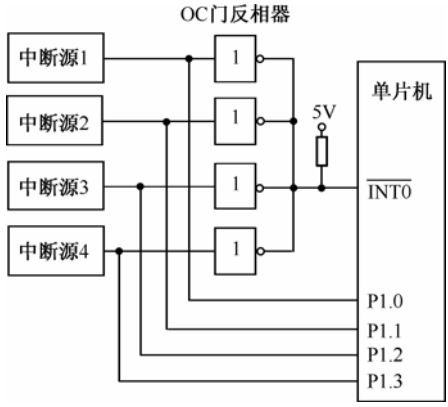


图 3.5　利用 OC 门的外部中断扩展电路

外部中断源查询的流程如图 3.6 所示，程序如下：

```
        ORG    0003H
        AJMP   EXINT0
MAIN:   …
EXINT0: PUSH   PSW          ；保护现场
        PUSH   ACC
        MOV    A，0FH
        MOV    P1，A
        MOV    A，P1        ；读入中断源
        RRC    A
        JC     EINT1        ；P1.0 为高电平转扩展中断 1 处理
        RRC    A
        JC     EINT2        ；P1.1 为高电平转扩展中断 2 处理
        RRC    A
        JC     EINT3        ；P1.2 为高电平转扩展中断 3 处理
        RRC    A
        JC     EINT4        ；P1.3 为高电平转扩展中断 4 处理
INTEND: POP        ACC      ；恢复现场
        POP    PSW
        RETI
EINT1:  …                   ；扩展中断 1 处理程序
        AJMP   INTEND
EINT2:      …               ；扩展中断 2 处理程序
        AJMP   INTEND
EINT3:  …                   ；扩展中断 3 处理程序
        AJMP   INTEND
EINT4:      …               ；扩展中断 4 处理程序
        AJMP   INTEND
```

请读者思考：如果单片机正在处理一个扩展中断源，又有另一个扩展中断源产生中断请求，请问这个中断请求是否会得到响应？怎样考虑它们的中断嵌套问题？又假设两个以上的

扩展中断源同时提出中断请求，怎样保证这些中断请求都得到响应？

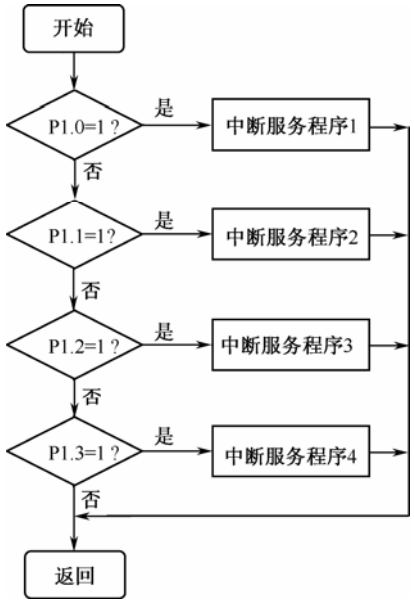


图 3.6　外部中断源查询的流程图

3.7　中断应用注意事项

（1）中断服务子程序与普通子程序虽然都是子程序，但它们之间有较大的差别。首先，调用者不同，普通子程序是用子程序调用指令完成调用，是由软件进行调用的。中断服务子程序是由 CPU 的中断系统调用的，调用是由硬件完成的。

（2）中断服务程序与普通子程序的调用时机不同，普通子程序的调用是在程序的固定位置，一旦程序写好就不再改变。而中断服务子程序是在中断系统响应中断请求时调用的，由于中断请求的随机性，因此中断服务子程序的调用也是随机的。

（3）普通服务子程序存放的位置可以随意安排，而中断服务子程序为了能与对应的中断源匹配，它的首次入口地址是固定的。

（4）中断服务子程序在返回时使用的是 RETI 指令，而普通子程序在程序末尾使用 RET 指令，RETI 指令除了弹出断点外，还将清除中断响应时所置位的优先级状态触发器，使得已申请的较低级中断请求可以响应。在实际应用中，如果在中断返回时误用了 RET 指令，可能会造成中断只响应一次的现象。

（5）由于 MCS-51 单片机没有停机指令，一旦工作就在不停地执行指令，在完成系统初始化后，循环完成某些工作，如图 3.7 所示。中断一般发生在单片机完成某些工作期间，这就意味着主程序可以随时被中断程序打断，因此主程序的优先级低于任何中断服务程序，应该在主程序中安排一些“不太重要”的工作。另外，一定注意中断服务子程序是由 CPU 的中断系统调用的，不要在主程序中进行中断服务子程序的调用。

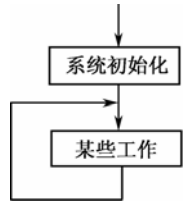


图 3.7　单片机系统主程序流程图

（6）由于中断发生是随机的，因而使得由中断驱动的中断服务程序难以把握，不便于调

试，这就要求在设计中断和中断服务程序时应特别谨慎，力求正确。对于中断服务程序比较庞大的情况，可以将程序的主要功能单独调试好后，再仔细转变为中断服务程序，在中断服务程序中尤其要注意参数的转递方式和途径，不能与其他程序发生冲突。

（7）由于从中断源发出中断请求信号到得到中断响应和处理需要一定的时间，且这个时间是不固定的，因此，在输入输出数据处理频度很高或实时处理的要求很高时，不宜采用中断方式，应该考虑其他有效办法。

本 章 小 结

中断是一个过程，是 CPU 暂停现行操作，去为提出服务要求的外设服务，当服务完成后回到原有操作继续执行的过程，中断是计算机的一项重要技术。本章要求：了解中断源、中断的功能、中断系统的组成与功能、中断优先权管理、中断响应条件、中断响应过程、中断嵌套等基本概念；掌握 MCS-51 单片机的 5 个中断源；掌握 MCS-51 单片机中断请求、中断允许、中断优先权管理特殊功能寄存器及其应用；掌握外部中断的边沿和电平触发方式，弄清各中断源中断请求信号的撤除方法；掌握 MCS-51 单片机的中断响应条件和响应过程，能根据实际需求，对中断系统进行初始化并编写简单的中断服务程序。

习 题 3

3.1 什么是中断、中断源、中断断点和中断响应？

3.2 MCS-51 单片机有哪些中断源，对应中断服务程序入口地址是什么？

3.3 MCS-51 单片机响应某一个中断请求的条件是什么？

3.4 如何控制 MCS-51 单片机的各中断源的使能和优先权？假如只禁止串口中断，选择外部中断 0 和定时/计数器 0 为高优先权，请确定特殊功能寄存器 IE 和 IP 的取值，并写出相应的程序段。

3.5 MCS-51 单片机的两个外部中断源有哪两种触发方式？不同触发方式下的中断请求标志是如何清 0 的？

3.6 MCS-51 单片机的中断响应过程是什么？

3.7 请叙述中断服务程序设计的一般格式。在什么情况下，中断服务程序要进行现场保护和现场恢复？怎样实现现场保护和现场恢复？现场保护和现场恢复应该放在中断服务程序的什么位置？

第 4 章　MCS-51 系列单片机定时器/计数器

本章主要内容

MCS-51 单片机的定时器/计数器的结构及工作原理。MCS-51 单片机定时器/计数器的工作方式及应用

定时器/计数器是单片机的重要功能模块，可实现定时控制、延时、对外部事件计数和检测等功能。在工业检测、自动控制以及智能仪器等方面起着重要作用。在第 2 章的学习中，我们知道，延时可以通过延时程序来实现，但延迟程序是利用 CPU 空转等待来实现的，会降低 CPU 的工作效率，且延时程序会受到中断的影响，延时时间并不可靠。而定时器/计数器是专门的计时/计数硬件，独立于 CPU 工作。目前各种型号的单片机，不管其功能强弱都提供有定时器/计数器，定时器主要完成系统运行中的定时功能，而计数器主要用于对外部事件的计数。

MCS-51 系列单片机的 51 子系列有两个 16 位定时器/计数器 T0、T1，而 52 子系列则有 3 个 16 位定时器/计数器 T0、T1、T2。本书只学习 T0、T1 的结构、功能及其应用。

4.1　MCS-51 系列单片机定时器/计数器的结构

51 子系列单片机内部有两个 16 位定时器/计数器：定时器/计数器 T0 和定时器/计数器 T1。它们都具有定时和计数功能，可用于定时或延时控制、对外部事件检测、计数等，其内部结构框图如图 4.1 所示。

MCS-51 的定时器/计数器的核心是一个加 1 计数器，也有单片机用的是减 1 计数器。定时器与计数器的区别主要在于计数脉冲的来源不一样。定时器的计数脉冲来源于时钟脉冲，而计数器的计数脉冲来源于代表外部事件的脉冲。任意一个定时器/计数器在定时时间到或计数到时，会产生相应的触发信号或中断信号以通知 CPU 进行相应处理。

由图 4.1 知，定时器/计数器 T0、T1 主要由存放计数初值和经过值的两对 8 位寄存器（TH0、TL0 和 TH1、TL1）、方式寄存器 TMOD 和控制寄存器 TCON 组成。

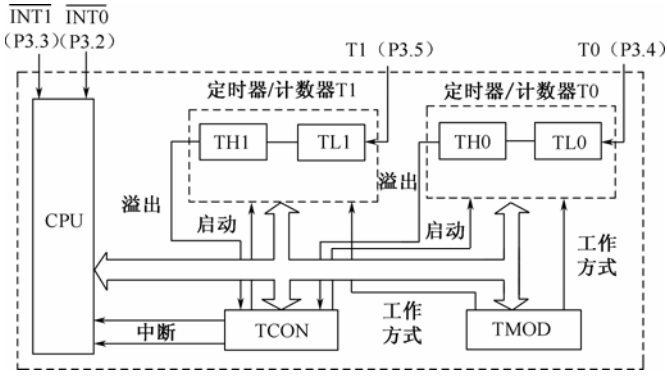


图 4.1　定时器/计数器结构框图

其中，TH0、TL0 专门用于存放定时器/计数器 T0 的计数初值和经过值；

TH1、TL1 专门用于存放定时器/计数器 T1 的计数初值和经过值；

TMOD 专门用于设置 T0、T1 的工作方式；

TCON 专门用于控制 T0、T1 的运行；

P3.4、P3.5 专门用于输入在计数器方式下的外部计数脉冲信号。

定时器/计数器 T0、T1 用作定时器时，加 1 计数器对机器周期进行计数，每过一个机器周期计数器加 1，直到计数器计满溢出。由于一个机器周期由 12 个时钟周期组成，所以计数频率为时钟频率的 1/12，比如当单片机采用 12MHz 晶振时，一个机器周期是 1μs，计 100 个数就相当于定时 100μs。所以定时器的定时时间不仅与计数器的初值即计数长度有关，而且还与系统的时钟频率大小有关。

定时器/计数器 T0、T1 用作计数器时，加 1 计数器对来自输入引脚 T0（P3.4）和 T1（P3.5）的外部信号计数。计数器对外部脉冲信号的占空比没有特别的要求，但必须保证输入的高电平和低电平信号至少应维持在一个完整的机器周期内保持不变。

4.2 MCS-51 系列单片机定时器/计数器的控制

MCS-51 系列单片机的定时器/计数器均是可编程控制的，在利用定时器/计数器进行定时或计数之前，必须要通过软件对它们进行初始化。初始化包括的内容如下：

（1）根据实际需要确定工作方式，对方式寄存器 TMOD 初始化。即根据实际的需要选择工作方式，形成相应的方式控制字，在程序中将方式控制字填充给 TMOD。

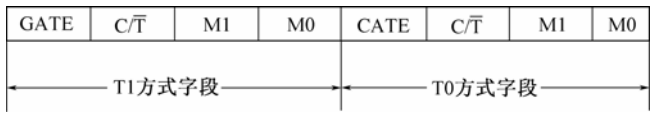
（2）根据实际定时/计数的需要确定初值，进行定时/计数初值的填充。

（3）根据需要开放相应的中断，即对中断允许控制寄存器 IE 进行初始化。

（4）启动定时器/计数器工作，即根据需要对 TCON 进行初始化。

4.2.1 定时器/计数器工作方式寄存器TMOD

特殊功能寄存器 TMOD 为 T0、T1 的工作方式寄存器，其映像地址为 89H，它只能按字节操作，复位时 TMOD 所有位均被清为 0。其格式如下：



TMOD 的低 4 位为 T0 的方式字段，高 4 位为 T1 的方式字段，它们的含义是完全相同的。

（1）M1M0：T0/T1 的工作方式选择位，其对应关系如表 4.1 所示。

表 4.1 T0/T1 工作方式选择位的意义

M1 M0	工作方式	功 能 说 明
0 0	方式 0	13 位定时器/计数器工作方式
0 1	方式 1	16 位定时器/计数器工作方式
1 0	方式 2	自动再装入的 8 位定时器/计数器工作方式
1 1	方式 3	T0：分为两个 8 位定时器/计数器，T1：停止计数

（2）C/$\overline{T}$：定时器或计数器方式选择位。当C/$\overline{T}$位为 0 时，选择定时器方式。在定时器

方式中，以时钟信号的 12 分频信号作为计数信号，也就是每一个机器周期定时器加 1。当系统的时钟信号频率确定后，定时器从初值开始加 1，到定时器溢出所需要的时间是固定的，所以称为定时器方式。

当 C/$\overline{T}$ 位置为 1 时，选择计数器方式。在计数器方式中，对外部引脚（T0 为 P3.4，T1 为 P3.5）上的输入脉冲信号进行计数。

（3）GATE：T0/T1 的门控位。当 GATE 位为 0 时，定时器/计数器 T0、T1 的运行仅受 TR0、TR1 的控制，不受外部引脚电平的状态的影响。

当 GATE 位置为 1 时，定时器/计数器 T0、T1 的运行不仅 TR0、TR1 的控制，而且还受到外部引脚电平状态的控制（$\overline{INT0}$ 控制 T0，$\overline{INT1}$ 控制 T1）。即只有当 $\overline{INT0}$（$\overline{INT1}$）引脚为高电平且 TR0（TR1）位为 1 时才启动 T0（T1）计数，当 $\overline{INT0}$（$\overline{INT1}$）引脚为低电平或 TR0（TR1）位为 0 时均使 T0（T1）停止计数。根据这一点，可以用 $\overline{INT0}$（$\overline{INT1}$）信号触发计数器，以此测量在 $\overline{INT0}$（$\overline{INT1}$）引脚上正脉冲的宽度。

例如，要使 T0 以方式 2 实现定时，T1 以方式 1 进行计数，且均与外部引脚电平无关的方式控制字为：01010010B，即 52H；若与外部引脚电平状态有关时的方式控制字为：11011010B，即 0DAH。

4.2.2 定时器/计数器控制寄存器TCON

特殊功能寄存器 TCON 的高 4 位为定时器/计数器 T0、T1 的运行控制位和计数溢出标志位，低 4 位为外部中断的触发方式控制位和锁存外部中断的请求标志位，其字节映像地址为 88H，可以按位操作。其格式如下：

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT1

TR0：定时器/计数器 T0 的启动标志位。当 TR0 位为 0 时，不允许 T0 计数工作； 当 TR0 位为 1 时，允许 T0 定时或计数工作。

TF0：定时器/计数器 T0 的溢出中断请求标标志位。在定时器/计数器 T0 被允许计数后，从初值开始加 1 计数，当产生计数溢出时由硬件自动将 TF0 位置为 1，通过 TF0 位向 CPU 申请中断，一直保持到 CPU 响应该中断后才由硬件自动将 TF0 位清为 0。当 TF0 位为 0 时，表示 T0 未计数或计数未产生溢出。当 T0 工作在不允许中断时，TF0 标志可供程序查询。

TR1：定时器/计数器 T1 的启动标志位。功能与 TR0 相同。

TF1：定时器/计数器 T1 的溢出中断请求标志位。功能与 TF0 相同。

IE0、IT0、IE1、IT1 的功能和作用已在 3.2.1 中介绍，在此不再重复。

4.3 定时器/计数器的工作方式及应用

MCS-51 单片机定时器/计数器有 4 种工作方式，不同的工作方式各有特点，其内部的结构有所不同，功能上也有差别。要用好定时器/计数器就必须对其进行深入了解。提请注意的是，只有定时器/计数器 T0 有 4 种工作方式，而定时器/计数器 T1 仅有 3 种工作方式。

4.3.1 定时器/计数器方式 0 及应用

方式 0 为 13 位定时器/计数器工作方式，方式 0 的内部结构如图 4.2 所示。

方式 0 由 TL0/TL1 的低 5 位和 TH0/TH1 的 8 位组成（而不是人们习惯的低 8 位和高 5 位！初学者容易弄错）。当 TL0/TL1 的低 5 位产生溢出进位时向 TH0/TH1 进位，TH0/TH1 计数溢出时置溢出中断请求标志位 TF0/TF1 为 1，向 CPU 请求中断。

由图 4.2 可知，在方式 0 的 T0/T1 计数脉冲控制电路中，有一个方式电子开关和允许计数控制电子开关。当 C/$\overline{T}$ 位为 0 时，方式电子开关与上面接通，以时钟频率的 12 分频信号作为 T0/T1 的计数信号；当 C/$\overline{T}$ 位为 1 时，方式电子开关与下面接通，此时以 T0（P3.4）/T1（P3.5）引脚上的输入脉冲作为 T0/T1 的计数脉冲。当 GATE 位为 0 时，由 TR0/TR1 控制定时器工作，当 GATE 位为 1 时，定时器不仅受 TR0/TR1 的控制，而且还受 $\overline{INT0}$ / $\overline{INT1}$ 引脚上的电平控制。

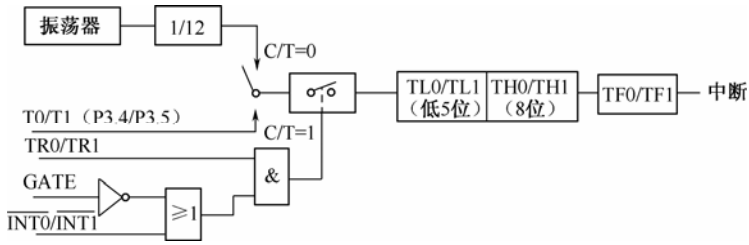


图 4.2　方式 0 的内部结构图

T0/T1 以方式 0 工作时定时/计数初值的计算方法：

当 T0/T1 以方式 0 定时工作时，假设时钟频率以 f_{osc} 表示，定时初值以 a 表示，定时时间以 t 表示，那么三者关系为：

$$t = \frac{12}{f_{osc}}(2^{13} - \alpha) \tag{4-1}$$

若给定定时时间的要求，定时初值 a 的大小为：

$$a = \frac{12 \times 2^{13} - t \times f_{osc}}{12} \tag{4-2}$$

例如，f_{osc}=12MHz，T0 的定时时间 t=5ms，则定时初值 a 为：

$$a = \frac{12 \times 2^{13} - 5 \times 10^{-3} \times 12 \times 10^{6}}{12} = 8192 - 5000 = 3192 = 01100011\ 11000B$$

特别要注意的是，在给定时器赋初值时将高 8 位赋给 TH，而将低 5 位赋给 TL 的低 5 位，因此，TL0 的初值为 18H，TH0 的初值为 63H，对 T0 的初始化的子程序为：

```
INTT0:  MOV   THO，#63H
        MOV   TL0，#18H
        SETB  EA
        SETB  ET0
        SETB  TR0
```

当 T0/T1 以方式 0 计数时，假设系统所需计取脉冲的个数以 x 表示，计数初值以 a 表示，则二者的关系为：

$$a = 2^{13} - x \tag{4-3}$$

例 4.1　利用 T0 产生 1ms 的定时，使 P1.0 引脚输出一个周期为 2ms，占空比为 1：1 的方波信号（假定时钟频率 f_{osc}=12MHz）。

解：分析要形成周期为 2ms 的方波信号，只要利用 T0 的定时每隔 1ms 将 P1.0 引脚状态

取反即可。

（1）确定工作方式：T0 要实现 1ms 的定时，方式 0 和方式 1 均可，在此选择方式 0；而在此题中没有对 T1 提出任何要求，所以 TMOD 的高 4 位任意。因此方式控制字为：

00000000B=00H

（2）确定定时初值：

$$a=\frac{12\times 2^{13}-1\times 10^{-3}\times 12\times 10^{6}}{12}=8192-1000=7192$$

即 a=11100000 11000B，高 8 位为 0E0H，低 5 位为 18H。

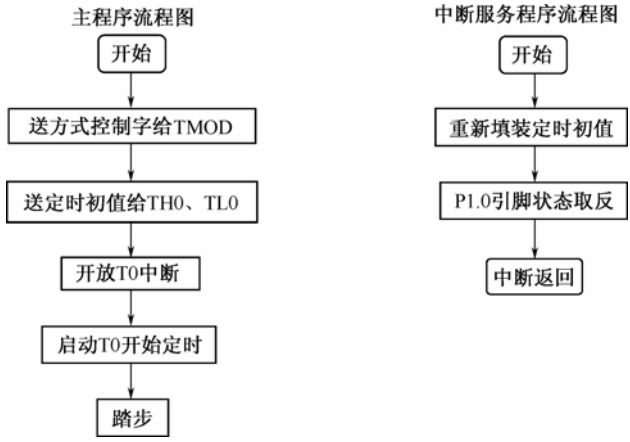


图 4.3　例 4.1 的控制程序流程图

（3）程序流程序图如图 4.3 所示，程序如下：

```
        ORG    0000H
        LJMP   MAIN
        ORG    000BH
        LJMP   INTT0
MAIN:   MOV    TMOD，#00H    ；送方式控制字
        MOV    TLO，#18H     ；送定时初值
        MOV    TH0，#0E0H
        SETB   EA            ；开放 T0 中断
        SETB   ET0
        SETB   TR0           ；启动 T0 定时
        SJMP   $             ；等待
INTT0:  MOV    TL0，#18H     ；重赋初值
        MOV    TH0，#0E0H
        CPL    P1.0          ；取反
        RETI
        END
```

方式 0 没有特别可取之处，它只是考虑到同 MCS-51 系列单片机的前辈 MCS-48 单片机兼容，才在定时器/计数器中设置这样一种工作方式，在 51 系列单片机的应用中几乎不再使用这种方式了。

4.3.2 定时器/计数器方式 1 及应用

方式 1 为 16 位定时器/计数器工作方式，方式 1 的内部结构如图 4.4 所示。

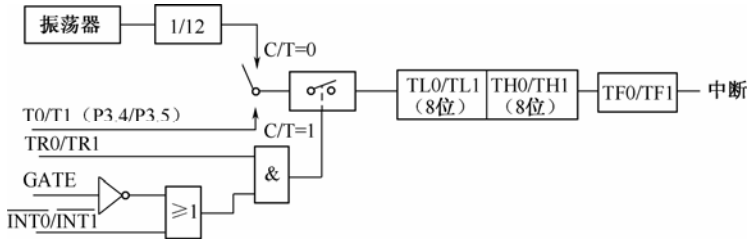


图 4.4　方式 1 的内部结构图

方式 1 由 TL0/TL1 的 8 位和 TH0/TH1 的 8 位组成。当 TL0/TL1 的 8 位产生溢出进位时向 TH0/TH1 进位，TH0/TH1 计数溢出时置溢出中断请求标志位 TF0/TF1 为 1，向 CPU 请求中断。

由图 4.4 可知，方式 1 的 T0/T1 计数脉冲控制电路与它们方式 0 的情况相似，仅仅是计数器的位数不同而已。

T0/T1 以方式 1 工作时定时/计数初值的计算方法：

当 T0/T1 以方式 1 定时工作时，假设时钟频率以 f_{osc} 表示，定时初值以 a 表示，定时时间以 t 表示，那么三者关系为：

$$t = \frac{12}{f_{osc}}(2^{16} - a) \tag{4-4}$$

若给定定时时间的要求，定时初值 a 的大小为

$$a = \frac{12 \times 2^{16} - t \times f_{osc}}{12} \tag{4-5}$$

例如，f_{osc}=12MHz，T1 的定时时间 t=20ms，则定时初值 a 为：

$$a = \frac{12 \times 2^{16} - 20 \times 10^{-3} \times 12 \times 10^{6}}{12} = 65536 - 2000 = 45536 = \text{B1E0H}$$

因此 TL1 的初值为 0E0H，TH1 的初值为 0B1H，对 T1 的初始化的子程序为：

```
INTT0:  MOV   TH1，#0B1H
        MOV   TL1，#0E0H
        SETB  EA
        SETB  ET1
        SETB  TR1
```

当 T0/T1 以方式 1 计数时，假设系统所需计取脉冲的个数以 x 表示，计数初值以 a 表示，则二者的关系为：

$$a = 2^{16} - x \tag{4-6}$$

例 4.2　采用定时器控制 P1.0 引脚输出一个周期为 2 分钟，占空比为 1∶1 的方波信号（时钟频率为 12MHz）。

解：要形成周期为 2 分种的方波信号，只要每隔 1 分种将 P1.0 引脚的状态取反即可。而 T0/T1 方式 1 不可能直接实现 1 分种的定时时间。对于这种长时间的定时，一般可以采用两种方法：一种方法是采用定时器定时加软件计数方法来实现，这种方法硬件资源省。另一种

方法是将两个定时器/计数器串联起来实现，其中一个定时器/计数器工作于定时器方式，另一个定时器/计数器工作于计数器方式，定时器的溢出信号作为计数器的计数脉冲输入信号，这种方法硬件资源占用多，而且要进行外部连线。

由于 MCS-51 系列单片机的定时器/计数器没有计数溢出输出引脚，要实现定时器/计数器级联需要通过其他引脚来实现，不仅硬件资源占用多，而且软件编程也不省事，所以通常不采用硬件级联这种方法。下面采用定时器加软件计数器的方法来实现长定时功能。

假设用 T1 定时 50ms，软件计数 1200 次，以此实现 1 分钟定时。为此用 20H 单元作 50ms 计数单元，计满 20 次为 1 秒；21H 单元为秒计数单元，计满 60 次为 1 分钟。

（1）方式控制字：要实现 50ms 的定时只有 T1 工作于方式 1，所以方式控制字为：

00010000B=10H

（2）定时初值。

$$a=\frac{12\times 2^{16}-50\times 10^{-3}\times 12\times 10^{6}}{12}=65536-5000=15536$$

即 a=11110010110000B=3CB0H

（3）程序流程图如图 4.5 所示，程序如下：

```
        ORG   0000H
        LJMP  MAIN
        ORG   001BH
        LJMP  INTT1
MAIN:   MOV   SP，#70H        ；设置堆栈区
        MOV   TMOD，#10H      ；送方式控制字
        MOV   TH1，#3CH       ；送定时初值
        MOV   TL1，#0B0H
        SETB  EA              ；开放 T1 中断
        SETB  ET1
        SETB  TR1             ；启动 T1 开始定时
        MOV   20H，#20        ；秒单元赋初值
        MOV   21H，#60        ；分单元赋初值
        SJMP  $               ；等待
INTT1:  MOV   TH1，#3CH       ；重赋初值
        MOV   TL1，#0B0H
        DJNZ  20H，NO         ；一秒钟未到，转 NO
        MOV   20H，#20
        DJNZ  21H，NO         ；一分钟未到，转 NO
        MOV   21H，#60
        CPL   P1.0
NO:     RETI
        END
```

无论是方式 0 还是方式 1，要实现重复定时或计数，都需要重新装载计数初值。在前面的例子中，中断服务程序一开始就进行计数器初值重装等操作。我们分析一下，在定时器溢出发出中断请求到重装完初值并在此基础上重新开始计数，总有一定的时间间隔，因此会造

成定时时间多增加了若干微秒的时间。另外，中断响应本身也需要一点时间，请读者估算一下，定时误差大约是多少？

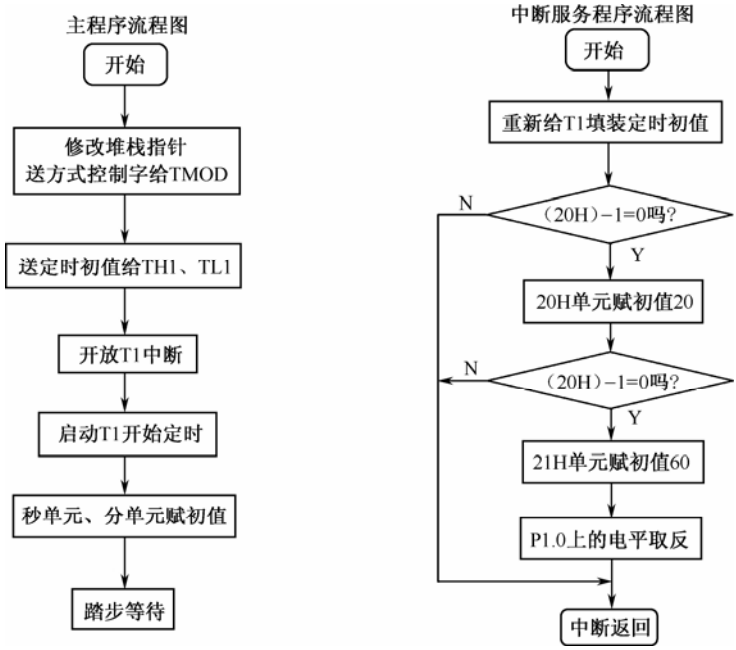


图 4.5　例 4.2 的控制程序流程图

4.3.3　定时器/计数器方式 2 及应用

方式 2 为自动恢复计数初值的 8 位定时器/计数器工作方式。T0/T1 工作于方式 0 或方式 1 时，若需要重复计数，就需要用户用指令重新填充初值；而方式 2 在计数器溢出时会自动地装入新的计数初值，开始新一轮计数。由于方式 0 或方式 1 是通过指令装入计数初值的，而执行指令需要时间，因此，方式 2 的定时时间比较准确。方式 2 的内部结构如图 4.6 所示。

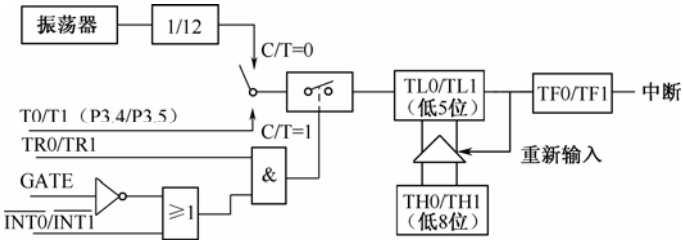


图 4.6　方式 2 的内部结构图

在方式 2 时，TL0/TL1 作为 8 位计数器，TH0/TH1 为自动恢复初值的 8 位计数器。当 TL0/TL1 计数发生溢出时，一方面置溢出中断请求标志 TF0/TF1 为 1，向 CPU 请求中断，同时又将 TH0/TH1 的内容送入 TL0/TL1，使 T0/T1 从初值开始重新加 1 计数。因此 T0/T1 工作于方式 2 时，定时精度高，但定时时间范围小。

由图 4.6 可知，方式 2 的 T0/T1 计数控制与方式 0 和方式 1 完全相同，不同之处在于当 CPU 响应 T0/T1 的溢出中断后会自动将 TH0/TH1 的内容填充到 TL0/TL1。

T0/T1 以方式 2 工作时定时计数初值的计算方法：

当 T0/T1 以方式 2 定时工作时，假设时钟频率以 f_{osc} 表示，定时初值以 a 表示，定时时间以 t 表示，那么三者关系为：

$$t = \frac{12}{f_{osc}}(2^8 - a) \tag{4-7}$$

若给定定时时间的要求，定时初值 a 的大小为：

$$a = \frac{12 \times 2^8 - t \times f_{osc}}{12} \tag{4-8}$$

例如，f_{osc}=12MHz，T0 的定时时间 t=200μs，则定时初值 a 为：

$$a = \frac{12 \times 2^8 - 200 \times 10^{-6} \times 12 \times 10^6}{12} = 256 - 200 = 56$$

因此，TL0 的初值为 56，TH0 的初值也为 56，对 T0 的初始化的子程序为：

```
INTT0:  MOV   THO，#56
        MOV   TL0，#56
        SETB  EA
        SETB  ET0
        SETB  TR0
```

当 T0/T1 以方式 0 计数时，假设系统所需计取脉冲的个数以 x 表示，计数初值以 a 表示，则二者的关系为：

$$a = 2^8 - x \tag{4-9}$$

例 4.3　将 T0 对外部脉冲进行计数，每计满 100 次后从 P1.0 输出宽度为 1ms 的高电平，如此循环下去（时钟频率为 6MHz）。

解：上述问题是 T0 交替工作于计数器方式和定时器方式，先计数，计满 100 次后改为定时器方式，1ms 后又回到计数器方式。设计数器为方式 2，定时器为方式 1，则

（1）T0 的方式控制字：计数时为 00000110B=06H；定时时为 00000001B=01H。

（2）T0 的定时初值：计数初值为 9CH；定时初值为 FE0CH。

程序流程图如图 4.7 所示。程序如下：

```
        ORG   0030H
CNT:    CLR   TR0
        MOV   TMOD，#06H    ；T0 为计数方式
        MOV   TH0，#9CH     ；送计数初值
        MOV   TL0，#9CH
        SETB  TR0           ；启动 T0 计数
        CLR   P1.0          ；P1.0 为低电平
WAIT1:  JBC   TF0，TIM      ；100 次满则转为定时
        SJMP  WAIT1         ；100 次未满则等待
TIM:    CLR   TR0           ；停止 T0 工作
        SETB  P1.0          ；P1.0 为高电平
        MOV   TMOD，#01H    ；T0 为定时方式
        MOV   TH0，#0FEH    ；送定时初值
        MOV   TL0，#0CH
        SETB  TR0           ；启动 T0 定时
```

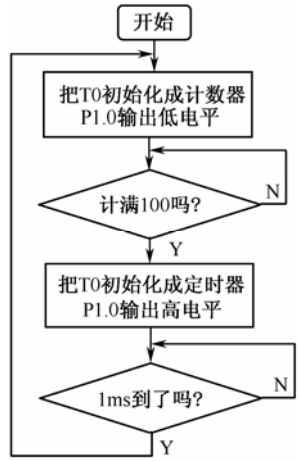


图 4.7　例 4.3 的控制程序流程图

```
WAIT2:  JBC     TF0，CNT        ；1ms 到则转为计数
        SJMP    WAIT2           ；1ms 未到则等待
        CLR     P1.0
        LJMP    CNT
        END
```

在上面这个例子中，定时器/计数器的溢出标志供程序查询，使用 JBC 指令，当查询到标志为 1 时，在实现转移的同时将溢出标志清除。请读者认真思考，若采用中断方式，而且要求只使用定时器/计数器 T0，该如何去实现呢？

4.3.4 定时器/计数器方式 3 及应用

方式 3 为两个 8 位定时器/计数器工作方式，它只适用于定时器/计数器 T0，这时 TH0 和 TL0 成为两个独立的 8 位定时器/计数器，使单片机增加了一个附加的 8 位定时/计数器。这对于资源有限的单片机系统有特别的意义。而 T1 不能工作于方式 3，若 T1 设置为方式 3，则 T1 将停止计数。方式 3 内部逻辑结构如图 4.8 所示。

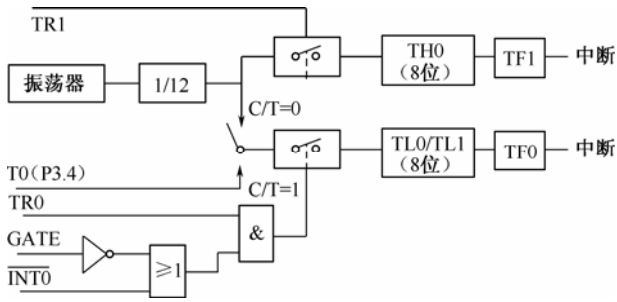


图 4.8　方式 3 的内部结构图

由图 4.8 可知，方式 3 是将 T0 拆分为两个 8 位定时器/计数器。其中：8 位计数器 TL0，使用原来 T0 的控制位（$C/\overline{T}$、GATE、TR0、$\overline{INT0}$）形成一个 8 位的定时器/计数器，它既可以定时（对时钟频率的 12 分频信号进行计数），也可以实现计数（对 P3.4 上的脉冲进行计数）；而 8 位计数器 TH0，占用原来 T1 的控制位（TR1、TF1），它只能对内部计数脉冲进行计数，其运行仅受 TR1 位的控制，而与外部引脚 $\overline{INT1}$ 电平状态无关。

定时器/计数器 T0 工作于方式 3 时，定时器/计数器 T1 仍可设置为方式 0、方式 1 和方式 2。但由于 TR1、TF1 已被定时器 TH0 占用，此时定时器/计数器 T1 仅由控制位 $C/\overline{T}$ 设置其定时或计数方式，当计数器计满发生溢出时，只能将溢出信号送往串行口。在这种情况下，T1 一般用作串行口波特率发生器，或不需要中断的场合。

方式 3 的两个 8 位定时器/计数器的定时或计数初值的计算方法与方式 2 完全相同，不再重复。

例 4.4　设某用户系统中已使用了两个外部中断源，并置定时器/计数器 T1 工作在模式 2，作为串行口波特率发生器。现要求再增加一个外部中断源，并由 P1.0 输出一个 5kHz 的方波。设单片机的振荡频率为 11.0592MHz。

解：根据要求，由 P1.0 输出一个 5kHz 的方波可由定时器实现，而需要增加的一个外部中断源可以通过计数器来模拟实现。由于单片资源的限制，可设置 T0 工作于模式 3，分成两个 8 位的定时器。TH0 工作于定时器方式，由它来实现 P1.0 上输出 5kHz 的方波；而将 TL0

工作于计数器方式，将 T0 引脚作为新增加的外部中断输入端，即将计数器的初值设为 0FFH，则只能接收一个脉冲，计 1 次数，因此当检测到 T0 引脚上出现由 1 至 0 的负跳变时，TL0 产生溢出，相当于产生外部中断。

TL0 的计数初值为：0FFH

TH0 的计数初值按如下方法计算：

方波的频率为 5kHz，则周期为 0.2ms，因此每半周期对 P1.0 取一次反就可获得要求的方波，故计数初值按 0.1ms 来计算，计数初值为：

$$a = 256 - \frac{100\times10^{-6}\times11.0592\times10^{6}}{12} = 256 - 92 = 164 = 0\text{A}4\text{H} \qquad (4\text{-}10)$$

定时器/计数器 T1 作为波特率发生器，工作于方式 2，根据波特率计算 T1 的计数初值，于是方式控制寄存器 TMOD 的取值为 27H。程序如下：

初始化程序段：

```
START:  MOV     TMOD，#27H      ；T0 为模式 3 计数，T1 为模式 2 定时
        MOV     TL0，#0FFH
        MOV     TH0，0A4H
        MOV     TL1，#baud      ；根据通信速率的要求设置
        MOV     TH1，#baud
        MOV     TCON，#55H      ；设定中断触发方式并启动 T0，T1
        MOV     IE，#9FH        ；开中断
          ⋮
```

TL0 溢出中断服务程序（由 000BH 转来）：

```
TL0INT: MOV     TL0，#0FFH
          ⋮
        RETI
```

TH0 溢出中断服务程序（由 001BH 转来）：

```
TH0INT: MOV     TH0，#0A4H      ；TH0 重新赋初值
        CPL     P1.0            ；输出方波
        RETI
```

4.3.5 定时器/计数器的其他应用举例

MCS-51 系列 单片机的定时器/计数器的功能十分强大，应用也非常灵活，既可定时，也能计数，既可查询，也能中断，还可扩充为外部中断源和波特率发生器。前面的应用举例，还没有使用到门控位 GATE，下面看一个利用定时器/计数器实现脉冲宽度测量的例子。

例 4.5 利用 T0 确定外部 $\overline{\text{INT0}}$ 引脚上出现的正脉冲的宽度，并将测到的机器周期的个数存入 30H 开始的单元中。

解：根据要求，将 T0 设定为定时器工作方式 1，门控位 GATE 置 1，初值取 0。当外部 $\overline{\text{INT0}}$ 引脚变为高电平时启动 T0 定时（外触发）；当外部 $\overline{\text{INT0}}$ 变为低电平时停止 T0 定时。这样 T0 停止计时时的当前值就是外部 $\overline{\text{INT0}}$ 引脚为高电平期间所经过的机器周期的个数（假设外部的脉冲宽度不超过 65535 个周期）。

（1）方式控制字为 00001001B=09H

（2）T0 的定时初值为 a=0000H

程序流程图如图 4.9 所示。程序如下：

```
        ORG   0030H
MAIN:   MOV   TMOD，#09H      ；送方式控制字
        MOV   TL0，#0         ；送定时初值
        MOV   TH0，#0
WAIT1:  JNB   P3.2，WAIT1     ；等待 P3.2 变为高电平
        SETB  TR0             ；启动 T0 开始定时
WAIT2:  JB    P3.2，WAIT2     ；等待 P3.2 变为低电平
        CLR   TR0
        MOV   R0，#30H        ；数据指针赋值
        MOV   @R0，TL0        ；保存机器周期数据的低 8 位
        INC   R0
        MOV   @R0，TH0        ；保存机器周期数据的高 8 位
        SJMP  $
        END
```

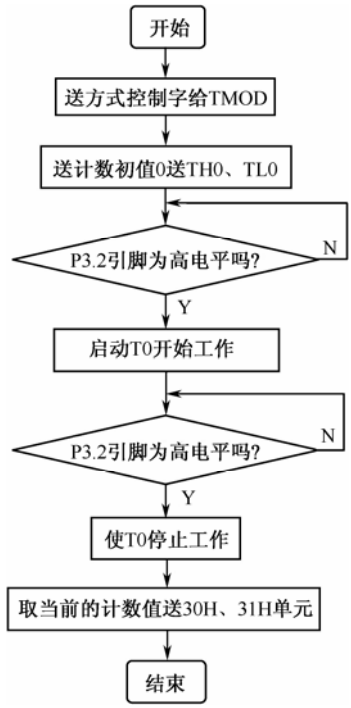


图 4.9　例 4.5 的控制程序流程图

请读者思考：如果要求既测量外部脉冲信号的高电平宽度，又要求测量外部脉冲信号的低电平宽度，该如何处理呢？

本 章 小 结

MCS-51 单片机有两个定时器/计数器 T0 和 T1，T0 具有 4 种工作方式，T1 只有 3 种工作方式。定时器

和计数器从本质上讲都是计数器，当其单片机内部的机器周期进行计数时，则称为定时器，因为一旦单片机系统时钟频率一定，则每计一次数的时间也是一定的。而当其对外部脉冲进行计数时，则称为计数器。定时时间和计数值可以通过程序设置。本章要求：掌握定时器/计数器的工作方式选择；掌握定时器/计数器各种工作方式及其计数初值计算；掌握定时器/计数器各种方式的初始化及其基本应用。

习 题 4

4.1 MCS-51 单片机的定时器/计数器的工作方式如何控制？如何启动 T0、T1 运行？

4.2 MCS-51 单片机的定时器/计数器 T0、T1 有哪些工作方式？它们的特点是什么？

4.3 定时器/计数器用作定时器时，其定时时间与哪些因素有关？作为计数器时，对外部计数脉冲的频率有何要求？

4.4 若时钟频率为 4MHz，则用 T0 产生 5ms 的定时，可以选择几种工作方式？分别写出各种工作方式下的方式控制字和定时初值。

4.5 利用 T0、T1 实现长定时的方法有哪些？它们各自的特点有哪些？试编写利用两个定时器/计数器合用实现长定时的控制程序。

4.5 设系统时钟为 6MHz，利用 T0 和 P1.0 产生连续矩形波，高电平宽度为 50μs，低电平宽度为 300μs。

4.6 设系统时钟为 12MHz，利用 T1 定时，编程从 P1.0 和 P1.1 引脚分别输出周期为 2ms 和 500μs 的方波。

4.7 让定时器/计数器 T1 对外部事件进行计数，每计数 1000 个脉冲后，让 T1 转为定时器工作方式，定时 10ms 后，再转为计数器方式，如此循环不止。设单片机振荡频率为 11.059MHz，请编程实现。

4.8 已知单片机的振荡频率为 6MHz，试编写程序，利用 T0 工作在方式 3，使 P1.0 和 P1.1 分别输出周期为 1ms 和 400μs 的方波。

第 5 章　MCS-51 系列单片机的扩展

本章主要内容

单片机三总线的形成方法。常用的程序存储器和数据存储器。存储器与单片机的连接方法以及地址空间的确定。I/O 接口扩展的方法及 8155、8255 的扩展应用。

MCS-51 单片机本身是一个基本的微型计算机，内部具有一定数量的存储单元和 I/O 接口，应用单片机时应充分利用内部已有的资源，当其资源不够用时才进行资源的扩充。本章就 MCS-51 系列单片机的存储器扩展、并行接口扩展等进行讨论，介绍一些常用的存储器芯片和 I/O 接口芯片，以及与单片机的连接和应用。

5.1　单片机三总线的形成及编址

5.1.1　单片机三总线的形成

所谓总线，就是连接系统中各扩展器件的一组公共信号线。按照功能，通常把系统总线分为三组，即地址总线、数据总线和控制总线。在 MCS-51 系列单片机中，P0 和 P2 口可以直接作为输入/输出口使用，也可以作为扩展总线口使用。在应用中，当单片机的内部资源不足时，就需要在外部增加芯片扩展相应的资源。为了使单片机能方便地与各种扩展芯片连接，应将单片机的外部连线形成一般计算机的三总线结构形式。

1. 地址总线（Address Bus,AB）

地址总线用于传送单片机送出的地址信号，以便进行存储单元和 I/O 端口的选择。地址总线是单向的，只能由单片机向外发送信息。地址总线的数目决定了可直接访问的存储单元或者端口的数目。如有 n 位地址则可以产生 2^n 个连续地址编码，因此，可访问 2^n 个存储单元或者端口，即通常所说的寻址范围为 2^n 个地址单元。当 MCS-51 单片机的 P0 口和 P2 口作为扩展总线口使用时，P0 口提供低 8 位地址信息，P2 口提供高 8 位地址信息，从而形成 16 位地址总线。因此，MCS-51 单片机可寻址 64KB 空间。

2. 数据总线（Data Bus,DB）

数据总线用于单片机与存储器之间或 I/O 端口之间传送数据。数据总线的位数与单片机处理数据的字长一致。MCS-51 单片机是 8 位字长计算机，所以数据总线的位数也是 8 位。数据总线是双向的，单片机可以通过数据总线将数据信息输出，也可以将外部的数据通过数据总线输入到单片机，但输入和输出不能同时进行。MCS-51 单片机的 P0 口作为地址/数据复用总线，当传输完地址信息后作为双向的 8 位数据总线，进行数据的传送。

3. 控制总线（Control Bus,CB）

控制总线是单片机发出的以控制片外 ROM、RAM 和 I/O 口读/写操作的一组控制线。MCS-51 单片机的 P3 口的一部分口线使用其第二功能，如 P3.6 和 P3.7 分别作为 $\overline{WR}$ 和 $\overline{RD}$ 信号对外部 RAM 或 I/O 口进行写/读控制。另外，$\overline{PSEN}$ 作为外部程序存储器的选通输出控制信号（相当于程序存储器空间读信号），ALE 输出地址锁存输出信号，用来分离通过 P0 口传输的地址和数据信号。

由此形成单片机的三总线：地址总线（P2、P0）、数据总线（P0）和控制总线（$\overline{RD}$、$\overline{WR}$、$\overline{PSEN}$、ALE 等），如图 5.1 所示。

需要说明的是，P0 口作为复用总线，地址和数据是分时传输的，先传输地址信号，ALE 脉冲的下降沿锁存地址信号，之后 P0 口可以传输数据信号了。在 MCS-51 单片机系统中，常用的地址锁存器是 74LS373，其引脚封装如图 5.2 所示。由于 P0、P2 口的驱动能力有限，当总线上连接的电路较多时，必须增加总线驱动电路，以增强总线的负载能力。

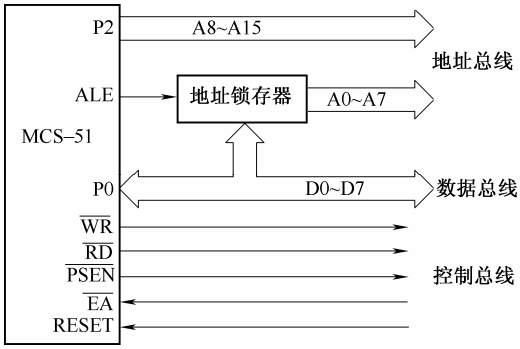


图 5.1　单片机的三总线结构示意图

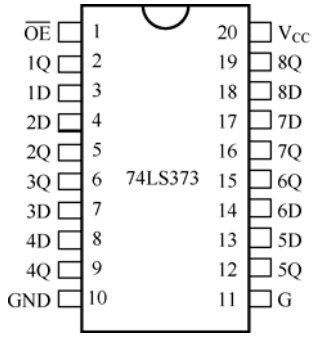


图 5.2　74LS373 的引脚封装图

地址锁存器 74LS373 在单片机扩展时起着十分关键的作用，它实现地址和数据的分离。74LS373 是一个带三态缓冲输出的 8D 锁存器，1D～8D 为数据锁存输入端，1Q～8Q 为锁存输出端，当 $\overline{OE}$ 为高电平时，锁存器处于高阻状态。当 $\overline{OE}$ 为低电平时，G 引脚为高电平期间，1Q～8Q 跟随 1D～8D 状态，而当 G 引脚电平由高跳变为低电平（下降沿）时，1Q～8Q 锁存数据，其状态不再随 1D～8D 变化而变化。在单片机应用系统中，$\overline{OE}$ 引脚通常接地，G 引脚接到单片机的 ALE 引脚上。

5.1.2　编址及译码

所谓编址，就是使用系统提供的地址总线，通过适当的连接确定一个存储单元（或 I/O 端口）的地址编号。

编址技术按类型分为两种：独立编址和统一编址。独立编址是指存储单元和 I/O 端口分开编址，在逻辑上是两个独立空间，分别使用不同的指令进行访问，如我们常用的 PC 机就是采用的这种编址方式。而统一编址是将存储单元和 I/O 端口在一个逻辑空间中统一编号，因此，操作存储单元和操作 I/O 端口没有本质差别，MCS-51 系列单片机就是采用的这种编址方式。

每一个存储单元或 I/O 端口有了地址编码后，要操作这些存储单元或 I/O 端口，就只需操作相应的地址单元即可。通过地址编码寻找到相应的存储单元或 I/O 端口叫译码，当单片

机的扩展资源不止一块集成电路时，首先要找到存储单元或 I/O 端口所在的芯片，称为片选；在存储芯片或 I/O 接口芯片中往往有若干存储单元或 I/O 端口，要区分这些单元或端口，就需要内部译码，称为字选。字选是通过存储芯片或 I/O 接口芯片内部译码来实现，而片选需要通过外部译码电路来解决，解决的方法主要有线选法和译码法。

1. 线选法

单片机在进行资源扩展时，通常不会扩展到极限的 64KB 空间，一般都会余下一定数量的地址信号（一般为 P2 口），利用这些地址线作为存储器或 I/O 接口芯片的片选信号，这种方法就叫做线选法。如图 5.3 所示，I、Ⅱ、III 都是 4KB×8 位存储器芯片，地址线 A11～A0 实现片内寻址，地址空间为 4KB。现用 3 根高位地址线 A14、A13、A12 实现片选，均为低电平有效。为了不出现寻址错误，A12、A13、A14 之中只能有一根地址线为低电平，也就是说每次存储器操作只能选中一个芯片，现假设剩下的 A15 为高电平，这样可得到 3 个芯片的地址分配，如表 5.1 所示。

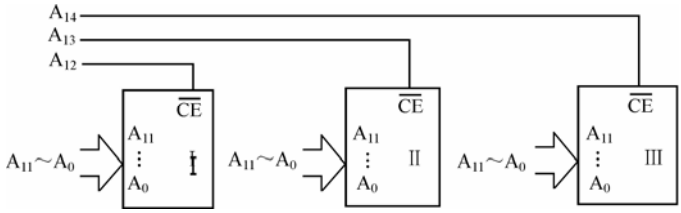


图 5.3　用线选法实现片选

表 5.1　线选法地址分配表

	二进制表示				十六进制
	A15 A14 A13 A12	A11		A0	
芯片 I	1 1 1 0	0 1	… …	0 1	E000H～EFFFH
芯片 II	1 1 0 1	0 1	… …	0 1	D000H～DFFFH
芯片 III	1 0 1 1	0 1	… …	0 1	B000H～BFFFH

从表 5.1 中看出 3 块芯片的内部寻址 A11～A0 都是 0～0（共 12 位）到 1～1（共 12 位），为 4KB 空间，而依靠不同的片选信号高位地址线 A14、A13、A12 之中某一根为 0，来区分这 3 个芯片的地址空间。

如果对两片存储器实现片选，可以用一根高位地址线加一“非门”实现，如图 5.4 所示。图中当 A12 为低电平时选通 I，当 A12 为高电平时选通Ⅱ，可得两芯片的地址空间为：

芯片 I　　E000H～EFFFH

芯片Ⅱ　　F000H～FFFFH

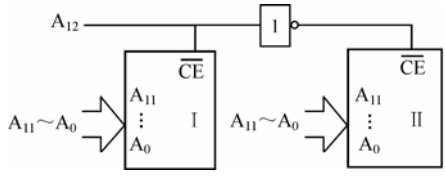


图 5.4　用一条高位地址线对两片 ROM 实现片选

从表 5.1 还不难看到，3 芯片的存储单元地址编号不连续，而且 A15 与它们没有关系，为 0 为 1 都是可以的，因此，线选法所确定的存储单元或 I/O 端口其地址编号也是不唯一的，各扩展芯片所占据的存储空间有重叠现象。比如图 5.3 所示电路，对于芯片 I，当 A15=0，则存储单元起始地址为 6000H，当 A15=1，则存储单元末尾地址为 EFFFH，因此，从 6000H 到 EFFFH 都是芯片 I 的存储空间。同样道理，对于芯片 II，它的地址空间范围是 5000H～DFFFH，不难看出，这两个存储空间具有重叠空间。线选法简单，它占用单片机硬件资源多，适用于不太复杂的系统。

2. 译码法

把扩展芯片内部用作字选的地址信号以外的地址信号全部通过译码器，所产生的译码信号作为芯片选择信号的方式，称为全译码方式，而将部分地址信号译码的方式称为部分译码方式。只有全译码方式所确定的存储单元或 I/O 端口，其地址编码是连续的、唯一的和不重叠的。译码法可以充分利用单片机的硬件资源，适合较复杂的大系统。地址译码法通常采用地址译码器，常用的地址译码器有 74LS138、74LS139、74LS154 等。

例如用译码法实现扩展容量 4KB×8 位的存储器芯片 I、Ⅱ、III 的接口电路，接线图如图 5.5 所示。地址线 A11～A0 用于片内寻址，高位地址线 A14、A13、A12 接到 74LS138 译码器的选择输入端 C、B、A，A15 作为译码器的高电平使能信号。74LS138 器的 $\overline{Y_2}$、$\overline{Y_1}$、$\overline{Y_0}$ 分别作为 3 个芯片的片选信号。

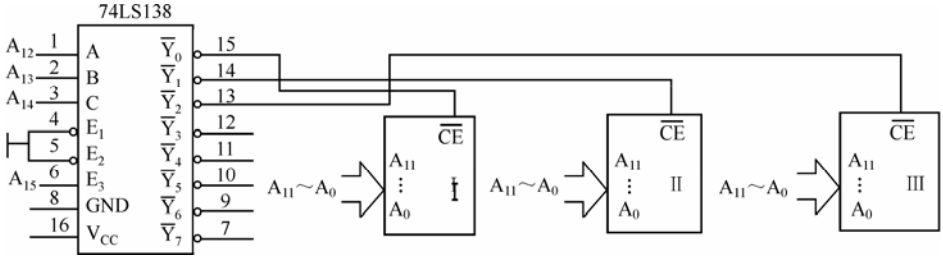


图 5.5　用全译码方式实现片选

根据译码器的逻辑关系和存储器的片内寻址范围，可以得到 3 个芯片地址空间，如表 5.2。

表 5.2　译码法实现片选的地址分配表

	二进制表示								十六进制表示
	A15	A14	A13	A12	A11	A10	…	A0	
芯片 I	1	0	0	0	0	0	…	0	8000H～8FFFH
	1	0	0	0	1	1	…	1	
芯片 II	1	0	0	1	0	0	…	0	9000H～9FFFH
	1	0	0	1	1	1	…	1	
芯片 III	1	0	1	0	0	0	…	0	A000H～AFFFH
	1	0	1	0	1	1	…	1	

从表 5.2 看出，由于除了扩展存储器所需的地址线外，剩余的全部地址线都参加译码，3 块芯片的地址空间连续，且唯一确定，不存在地址重叠现象，能够充分利用内存空间。

利用剩余地址线中的一部分地址线参加译码称为部分译码方式。部分译码有地址重叠现象，它兼有线选和全译码的某些特点，读者可自行分析。

5.2 存储器的扩展

MCS-51 单片机的程序存储器空间和数据存储器空间是相互独立的，程序存储器空间和数据存储器空间分别最大可扩展至 64KB。数据存储器和 I/O 端口是统一编址的，在 64KB 的外部 RAM 空间中，可划出一定区间作为外部扩展接口的地址空间。

5.2.1 程序存储器扩展

当前流行的单片机都有一定容量的内部程序存储器，早期的单片机有 EPROM 或 EEPROM 型程序存储器，随着技术的不断进步，FlashROM 型单片机大量出现并广泛应用，因此单片机系统一般很少扩展程序存储器。但对于无 ROM 型单片机（如早期的 8031、8032 等）或者单片机片内 ROM 不够用的情况下，构成系统时才需要扩展程序存储器。

1. 常用的程序存储器

外部程序存储器一般采用 EPROM 存储器、EEPROM 存储器和 FlashROM 存储器。

（1）EPROM：紫外线擦除电可编程只读存储器 EPROM 是国内用得较多的程序存储器。EPROM 芯片上有一个玻璃窗口，在紫外线照射下，存储器中的各位信息均变 1，即处于擦除状态。擦除干净的 EPROM 可以通过编程器将应用程序固化到芯片中。常用的 EPROM 有 2716、2732、2764、27128、27256、27512 等，引脚图如图 5.6 所示。

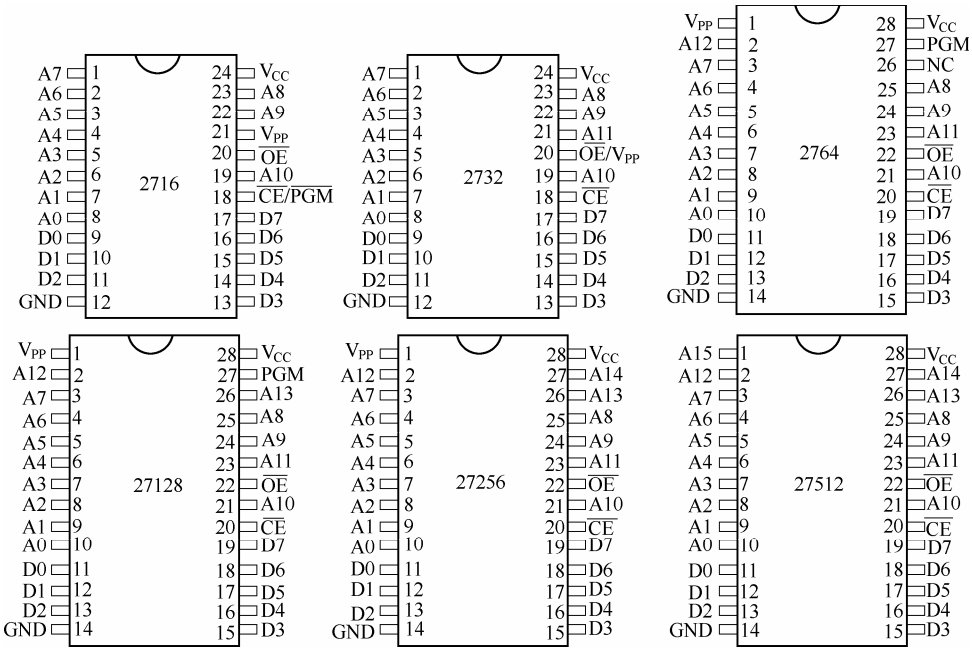


图 5.6 常用 EPROM 芯片引脚封装示意图

由图 5.6 可以看出，不同的 EPROM 芯片仅仅是地址线数目和编程信号引脚有些区别，各引脚的含义如下：

A0～Ai：地址输入线，i=11～15。

D0～D7：双向三态数据线。读时为输出线，编程时为输入线，禁止时为高阻（有些资料

中用 O0～O7 表示）。

$\overline{CE}$：片选信号输入线，低电平有效。

$\overline{OE}$：读选通输入信号线，低电平有效。

$\overline{PGM}$：编程脉冲信号输入线，是一个脉冲序列。

V_{PP}：编程电源输入线，V_{PP} 的幅值因芯片型号和制造厂商而异。

V_{CC}：芯片正常工作所需的电源输入线，一般为单一的+5V。

GND：芯片正常工作所需的接地线。

EPROM 芯片的主要操作方式有：编程方式、编程校验方式、读出方式、维持方式、编程禁止方式。EPROM 芯片在不同操作方式下控制引脚电平的状态如表 5.3 所示（以 2764 和 27256 为例），常用 EPROM 的主要技术特性如表 5.4 所示。

表 5.3　EPROM 操作方式选择表

		$\overline{CE}$	$\overline{OE}$	V_{PP}	V_{CC}	$\overline{OE}/V_{PP}$	$\overline{PGM}$	输　出
2764	引脚号	20	22	1	28		27	11～13，15～19
	读	V_{IL}	V_{IL}	V_{CC}	V_{CC}		V_{IH}	DOUT
	维持	V_{IH}	任意	V_{CC}	V_{CC}		任意	高阻
	编程	V_{IL}	V_{IH}	V_{PP}	V_{CC}		V_{IL}	DIN
	编程检验	V_{IL}	V_{IL}	V_{PP}	V_{CC}		V_{IH}	DOUT
	编程禁止	V_{IH}	任意	V_{PP}	V_{CC}		任意	高阻
27256	引脚号	20	22	1	28			11～13，15～19
	读	V_{IL}	V_{IL}	V_{CC}	V_{CC}			DOUT
	维持	V_{IH}	任意	V_{CC}	V_{CC}			高阻
	编程	V_{IL}	V_{IH}	V_{PP}	V_{CC}			DIN
	编程检验	V_{IL}	V_{IL}	V_{PP}	V_{CC}			DOUT
	编程禁止	V_{IH}	任意	V_{PP}	V_{CC}			高阻

表 5.4　常用 EPROM 的主要技术特性

型　　号	2716	2732	2764	27128	27256	27512
容量（KB）	2	4	8	16	32	64
引脚数	24	24	28	28	28	28
读出时间（ns）	350～450	100～300	100～300	100～300	100～300	100～300
最大工作电流(mA)		100	75	100	100	125
最大维持电流(mA)		35	35	40	40	40

（2）EEPROM：EEPROM 是一种电擦除可编程只读存储器，其主要特点是能在计算机系统中进行在线修改和擦除，并能在断电的情况下保持修改的结果，其功能相当于磁盘。较新的 EEPROM 产品在写入时还能自动完成擦除，且不需要专用的编程电源，可以直接使用单片机系统的+5V 电源。因而在智能化仪器仪表、控制装置等领域得到普遍采用。

目前 EEPROM 品种很多，分串行 EEPROM 和并行 EEPROM，串行 EEPROM 常作为数据存储器，将在下一章讨论，在此只讨论并行 EEPROM 芯片。在图 5.7 中列出了 4 种 EEPROM 芯片的外部引脚示意图。

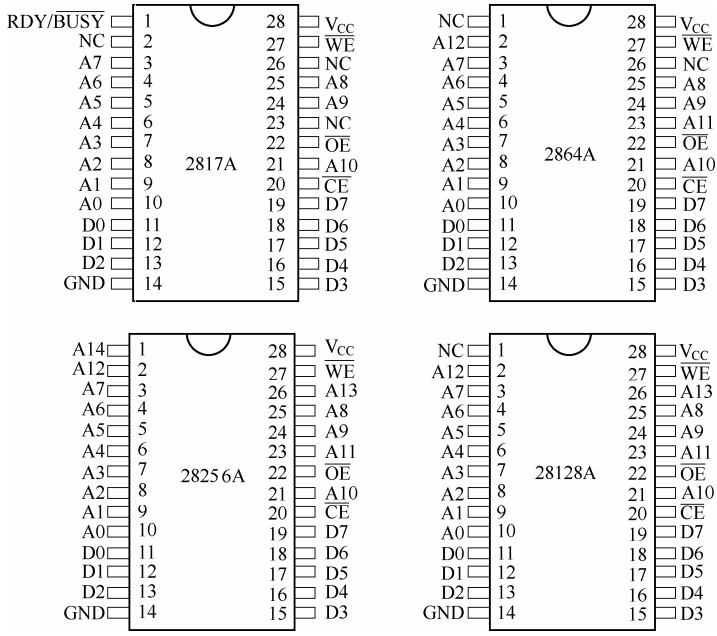


图 5.7　常用 EEPROM 芯片引脚封装示意图

由图 5.7 可知，不同的 EEPROM 芯片仅仅是地址线数目和编程信号引脚有些区别，各引脚的含义如下：

A0～Ai：地址输入线。

D0～D7：双向三态数据线，有时也经常用 I/O0～I/O7 表示。

$\overline{CE}$：片选信号输入线，低电平有效。

$\overline{OE}$：读选通信号输入线，低电平有效。

$\overline{WE}$：写选通信号输入线，低电平有效。

RDY/$\overline{BUSY}$：2817A 的状态输入线，低电平表示芯片正在进行写操作，写入后呈高阻状态；高电平表示芯片准备接收数据。

V_{CC}：工作电源，单一的+5V。

GND：芯片正常工作需要的接地线。

EEPROM 芯片的主要操作方式有：读方式、写方式、维持方式。EEPROM 芯片在不同操作方式下控制引脚电平的状态如表 5.5 所示（以 2817A 和 2864A 为例），芯片的主要技术特性如表 5.6 所示。

表 5.5　EEPROM 芯片的不同操作方式

		$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	RDY/$\overline{BUSY}$	I/O0～I/O7
2817A	引脚	20	22	27	1	11～13，15～19
	读	V_{IL}	V_{IL}	V_{IH}	高阻	数据输出
	写	V_{IL}	V_{IH}	V_{IL}	V_{IL}	数据输入
	维持	V_{IH}	任意	任意	高阻	高阻
2864A	引脚	20	22	27		11～13，15～19
	读	V_{IL}	V_{IL}	V_{IH}		数据输出
	写	V_{IL}	V_{IH}	V_{IL}		数据输入
	维持	V_{IH}	任意	任意		高阻

表 5.6　EEPROM 芯片的主要技术特性

型　　号	2817A	2864A	28128A	28256A
容量（KB）	2	8	16	32
引脚数	28	28	28	28
维持电流(mA)	55	60	75	75
工作电流(mA)	≤150	≤160	≤150	≤150
读出时间(ns)	250	250	250	250
信息保存时间（年）	10	10	10	10
最大存储时间（ms）	10～20	10	9～15	9～15

（3）FlashROM：FlashROM 是一种新型的电擦除式闪速存储器，属于 EEPROM 的改进产品。它的最大特点是必须按块擦除，而 EEPROM 则可以一次只擦除一个字节。只有在写入时，才以字节为最小单位写入。

FlashROM 品种繁多，如 29C256、 29C512、M29F010、M29F040 等。在图 5.8 中给出了两种 FlashROM 芯片的 PDIP 引脚封装图。

由图 5.8 可知 FlashROM 芯片的引脚含义如下：

A0～Ai：地址输入线。

I/O0～I/O7：双向三态数据线。读时为输出线，编程时为输入线，禁止时为高阻（有些资料中用 DQ0～DQ7 表示）。

$\overline{CE}$：片选信号输入线，低电平有效，有时用 $\overline{E}$ 表示。

$\overline{OE}$：读选通输入信号线，低电平有效，有时用 $\overline{G}$ 表示。

$\overline{WE}$：写选通信号输入线，低电平有效，有时用 $\overline{W}$ 表示。

V_{CC}：芯片正常工作所需的电源输入线，一般为单一的+5V。

GND：芯片正常工作所需的接地线，有时用 V_{SS} 表示。

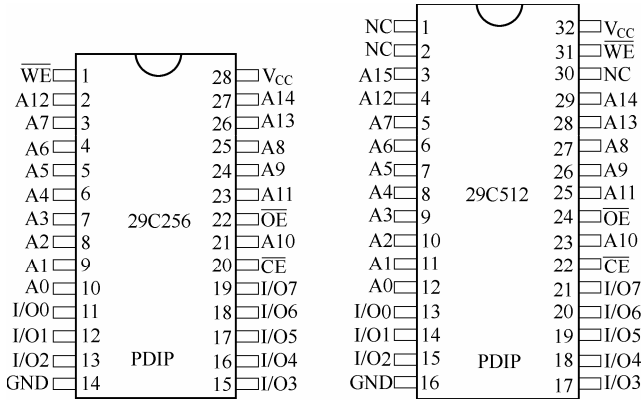


图 5.8　FlashROM 芯片引脚封装示意图

FlashROM 芯片在不同操作方式下控制引脚电平的状态如表 5.7 所示（以 29C256 为例），芯片的主要技术特性如表 5.8 所示。常用的 FlashROM 还有 28F×××系列，请读者查阅相关资料。

2. 程序存储器扩展实例

例 5.1　MCS-51 单片机与 27256 芯片的接口。

表 5.7　FlashROM 芯片的不同操作方式

		$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	I/ O0～I/O7
29C256	引脚	20	22	1	11～13，15～19
	读	VIL	VIL	VIH	数据输出
	写	VIL	VIH	VIL	数据输入
	维持	VIH	任意	任意	高阻

表 5.8　FlashROM 芯片的主要技术特性

型　　号	29C256	29C512
容量（KB）	32	64
引脚数	28	32
维持电流(μA)	300	100
工作电流(mA)	50	50
读出时间(ns)	70	70
信息保存时间（年）	15～20	15～20
最大存储时间（ms）	10	10

解：MCS-51 单片机与 27256 芯片的接口连线方法如图 5.9 所示，在图中 $\overline{OE}$ 用 $\overline{PSEN}$ 控制，$\overline{CE}$ 直接接地，27256 的 A0～A7 与 P0 口经 74LS373 锁存后 Q0～Q7 相连，A8～A14 分别与 P2.0～P2.6 相连，所以 27256 的 32KB 单元的基本地址空间为：0000H～7FFFH。

思考：上例中，为什么 $\overline{CE}$ 可以直接接地？重叠空间是多大？

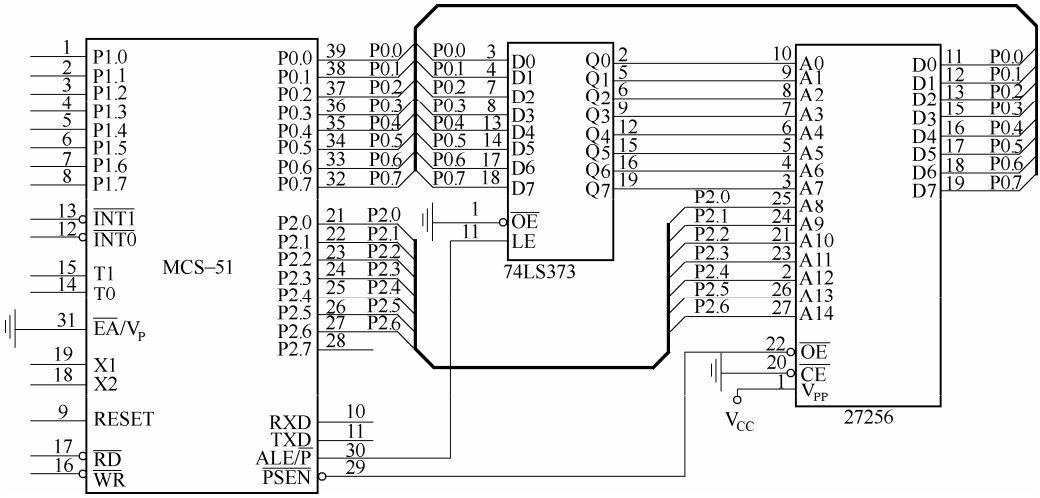


图 5.9　MCS-51 单片机与 27256 的接口连线图

程序存储单元通常是单片机取指令时被访问，但也可以通过指令 MOVC A，@A+PC 或 MOVC A，@A+DPTR 读取程序存储单元中的数据，即所谓查表。如读取上例 27256 EPROM 地址为 1000H 单元的内容的指令为

```
MOV     DPTR，#1000H
MOV     A，#00H
MOVC    A，@A+DPTR
```

例 5.2　MCS-51 单片机与 2864A 芯片的接口。

解：MCS-51 单片机与 2864A 芯片的接口电路如图 5.10 所示，由于 2864A 不仅可以存放程序，也可以在线写入数据，因此用与门将单片机的 $\overline{\text{PSEN}}$ 和 $\overline{\text{RD}}$ 信号合并作为 $\overline{\text{OE}}$，单片机的 $\overline{\text{WR}}$ 信号与 2864A 的 $\overline{\text{WE}}$ 信号直接相连，于是程序存储器空间和数据存储器空间就合二为一了。$\overline{\text{CE}}$ 用 P2.7 线选，则 2864A 的基本地址范围是 0000H～1FFFH。

思考：EEPROM 可以在线写入数据，试问它是否能像写 RAM 一样写入？

FlashROM 是快速擦写的存储器，它与 EEPROM 相似，与单片机的连接方法也相似，由于篇幅所限在此就不再举例了。

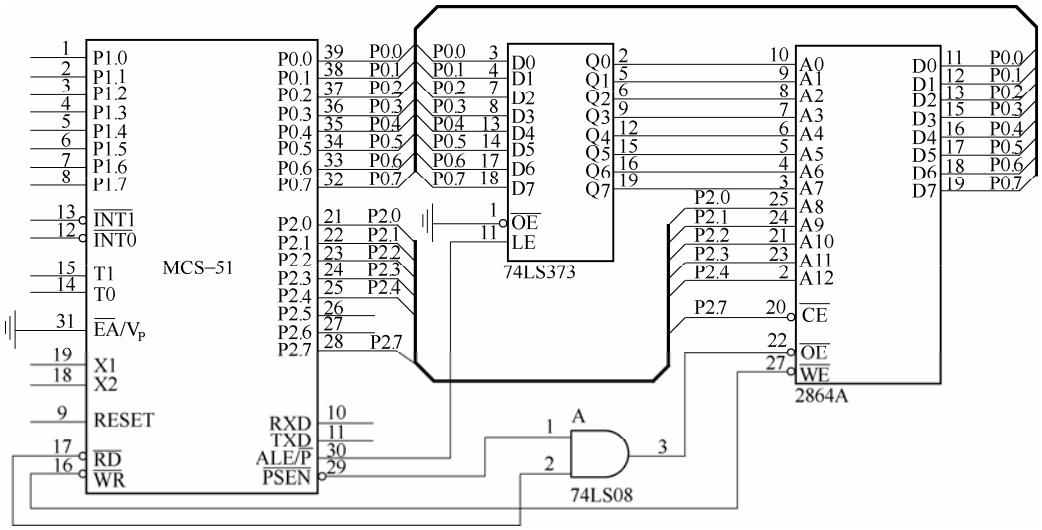


图 5.10　MCS-51 与 2864A 的接口电路图

5.2.2　数据存储器的扩展

MCS-51 系列单片机内部具有 128 或者 256 字节的 RAM 数据存储器，它们可以作为工作寄存器、堆栈、软件标志和数据缓冲器使用，CPU 对内部 RAM 具有丰富的操作指令。对于大多数控制系统来说，内部 RAM 空间可以满足系统对数据存储器的要求。而对于大容量数据处理的系统，在内部提供的 RAM 空间不足时，就需要在单片机外部扩展数据存储器。

1. 常用的数据存储器

数据存储器用于存储现场采集的原始数据、运算结果等，所以外部数据存储器应能随机地进行读或写。数据存储器可分为静态数据存储器 SRAM 和动态数据存储器 DRAM，虽然 DRAM 的集成度高，但需要刷新才能维持存储信息。对于单片机应用系统，数据存储器的容量小，不会因使用 DRAM 而设计刷新电路和刷新逻辑，通常采用 SRAM 作为外部数据存储器。SRAM 具有存取速度快、使用方便等优点，一旦掉电，其内部所存数据信息便会丢失。

常用的 SRAM 有 6116、6264、62256 等芯片。为了避免掉电丢失数据，近年来出现了掉电自动保护的静态 RAM，如 DS1225、DS1235 等，它们的引脚与 6264 和 62256 是兼容的。常用的典型静态数据存储器芯片引脚封装如图 5.11 所示，从图 5.11 可以看到，不同的静态 RAM 芯片仅仅是地址线数目和编程信号引脚有些区别，各引脚的含义如下：

A0～Ai：地址输入线，i=11～15。

O0～O7：双向三态数据线。读时为输出线，编程时为输入线，禁止时为高阻（有些资料

中用 DQ0～DQ7 表示）。

$\overline{CE}$：片选信号输入线，低电平有效。

$\overline{OE}$：读选通输入信号线，低电平有效。

$\overline{WE}$：写选能输入信号线，低电平有效。

V_{CC}：芯片正常工作所需的电源输入线，一般为单一的+5V。

GND：芯片正常工作所需的接地线。

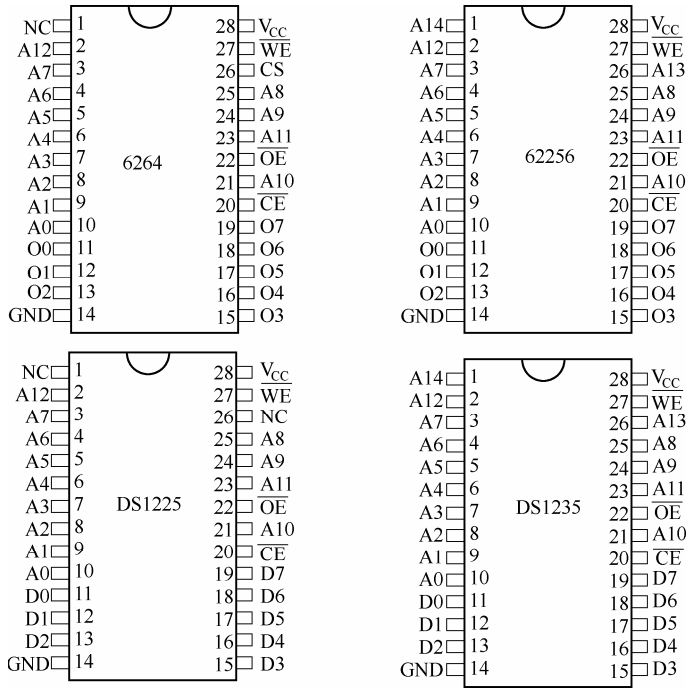


图 5.11　常用数据存储器芯片引脚封装示意图

静态 ARM 芯片不同操作方式下控制引脚电平的状态如表 5.9 所示，常用 RAM 的主要技术特性如表 5.10 所示。

表 5.9　静态 RAM 芯片不同操作方式

	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	O0～O7
读	VIL	VIL	VIH	数据输出
写	VIL	VIH	VIL	数据输入
维持*	VIH	任意	任意	高阻态

表 5.10　常用 RAM 芯片的主要技术特性

型　　号	DS1225	DS1235	6264	62256
容量（字节）	8K	32K	8K	32K
引脚数	28	28	28	28
工作电压（V）	5	5	5	5
典型工作电流（mA）	10	10	40	8
典型维持电流（μA）	200	200	2000	900
存取时间（ns）	由具体型号而定			

2. 扩展实例

例 5.3 MCS-51 单片机与 DS1235 的扩展电路。

解：MCS-51 系列单片机与并行的数据存储器芯片的接口方法与程序存储器扩展方法基本相同，数据存储器的读写信号使用 $\overline{RD}$ 和 $\overline{WR}$，不再使用 $\overline{PSEN}$。由于 MCS-51 单片机扩展的并行数据存储器与外部端口统一编址，共同占用 64KB 的地址空间，因此外部 RAM 的片选信号一般可由高位地址经译码产生，或用线选法直接作为片选信号。如图 5.12 所示，图中，用 P2.7 作为 DS1235 的片选信号，而 DS1235 的读写控制信号直接与 MCS-51 单片机的读写控制信号相连。要实现对 DS1235 单元的操作，首先必须保证片选信号 P2.7 为低电平，而地址线 A0～A14 可以任意变化，其取值情况无非是从 15 个“0”变化到 15 个“1”。所以 DS1235 片内 32KB 单元的地址空间为：0000000000000000B～0111111111111111B，即 0000H～7FFFH。

例如，从 DS1235 的地址为 40H 单元读取一个字节数据到累加器 ACC 中的程序为：

```
READB：MOV   DPTR，#0040H
       MOVX  A，@DPTR
```

上述功能也可以用如下方式实现：

```
READB：MOV   P2，#00H
       MOV   R0，#40H
       MOVX  A，@R0
```

由此可见，P2 可以通过 I/O 方式来输出地址，如果把 256 字节称为一页的话，则把 P2 口输出的地址称为页地址。与此类似，也可以把数据放入 DS1235 的某一单元。

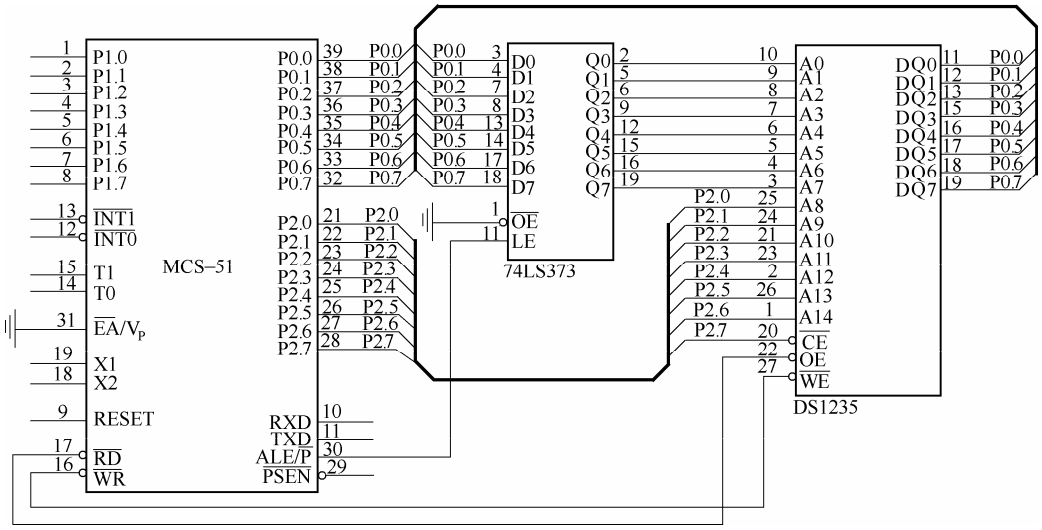


图 5.12 MCS-51 与 DS1235 的接口电路图

5.3 输入/输出口的扩展

MCS-51 系列单片机具有 4 个 8 位的 I/O 口（即 P0、P1、P2、P3），原则上 4 个口均可作为双向并行 I/O 接口。但在实际应用中，P0、P2 口常用作扩展总线，P3 口的一些口线又常使用其第二功能，特别是无 ROM 型单片机更是如此。所以如果一个 MCS-51 单片机应用系

统需要连接较多的并行输入输出的外围设备，就要扩展并行接口。在 51 单片机中扩展的 I/O 口采用与片外数据存储器相同的寻址方法，所有扩展的 I/O 口，以及通过扩展 I/O 口连接的外设都与片外 RAM 统一编址，因此，对片外 I/O 口的输入/输出指令就是访问片外 RAM 的指令。

常用的并行 I/O 接口扩展器件主要有简单 I/O 接口电路和可编程 I/O 接口电路两类。

5.3.1 简单接口芯片的扩展

当应用系统需要扩展的 I/O 口数量较少，且功能单一时，一般采用缓冲器和锁存器等集成电路芯片构成简单的 I/O 口。这种接口通过 P0 口进行扩展，由于 P0 口是数据/地址总线，因此扩展的输入口面向总线，必须是三态的；扩展的输出口连接外部设备，应具有锁存功能。因此扩展简单并行口选择器件的原则是“输入三态，输出锁存”。常用于扩展的缓冲器有 74LS244、74LS245 等；常用于扩展的锁存器有 74LS273、74LS373、74LS377、74LS573 等。由于简单接口芯片的品种多、价格低，是 MCS-51 单片机的并行 I/O 口扩展是常用的方法之一。下面介绍常用电路的接口方法。

1. 用缓冲器扩展输入口

（1）用 74LS244 进行扩展。74LS244 是一种史密特触发的 8 位三态缓冲器，抗干扰性好，它的引脚封装如图 5.13 所示。当它的控制端 $\overline{1G}$（$\overline{2G}$）为低电平时，输出等于输入（直通）；当 $\overline{1G}$（$\overline{2G}$）为高电平时，输出呈高阻态。

在图 5.14 所示的电路中，用 P2.7 与 $\overline{RD}$ 经或运算后作为 74LS244 的芯片使能（$\overline{1G}$，$\overline{2G}$），P0 口线与 74LS244 的 8 位输出端 1Y1～2Y4 相连，1A1～2A4 与外部设备相连，所以 74LS244 的访问地址为 7FFFH。通过 74LS244 读取外围设备数据到累加器 ACC 中的程序如下：

```
LS244:   MOV   DPTR，#7FFFH  ；将 74LS244 的口地址送入 DPTR
         MOVX  A，@DPTR      ；从 74LS244 端口读取数据到累加器 ACC
```

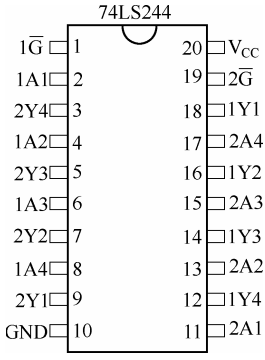


图 5.13　74LS244 的引脚封装图

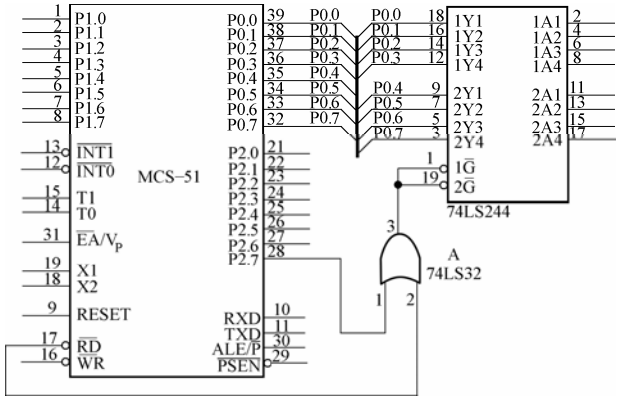


图 5.14　MCS-51 单片机与 74LS244 的连接图

（2）用 74LS245 进行扩展。74LS245 是一种三态双向缓冲器，常用作 8 位数据总线收发器/驱动器，也可用来扩展输入口。由于是双向的，8 位数据既可从 A 端传送到 B 端，也可从 B 端传送到 A 端（由方向控制 DIR 信号电平而定），还可以禁止传输（由使能信号 $\overline{G}$ 控制），74LS245 的引脚封装如图 5.15 所示。

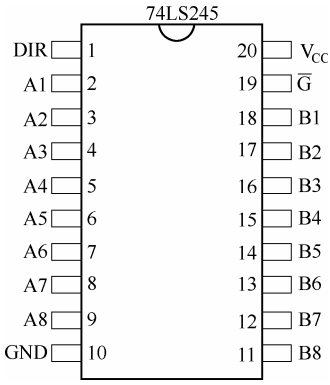


图 5.15　74LS245 的引脚封装图

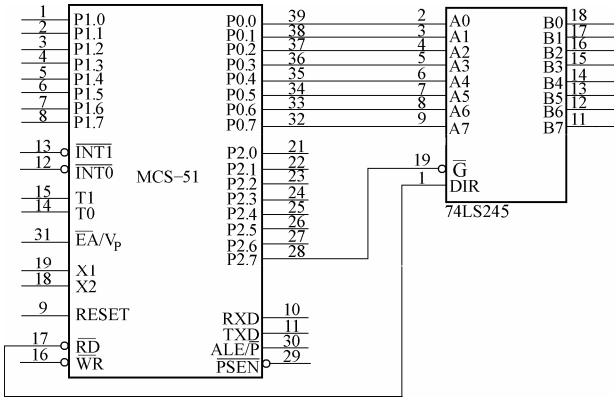


图 5.16　MCS-51 单片机与 74LS245 的接口电路图

在图 5.16 所示的电路中，用 P2.7 控制 74LS245 的使能端 $\overline{G}$（$\overline{E}$），用 $\overline{RD}$ 控制数据从 B 端到 A 端传送，P0 口线依次与 74LS245 的 A0～A7 相连，B0～B7 与外部设备的相连，所以 74LS245 的端口地址为 7FFFH。通过 74LS245 读取外围设备数据到累加器 ACC 中的程序如下：

```
LS245：  MOV   DPTR，#7FFFH      ；将 74LS245 的口地址送入 DPTR
         MOVX  A，@DPTR          ；从 74LS245 端口读取数据到累加器 ACC
```

2. 用锁存器扩展输出口

74LS273、74LS377 等均为 8D 锁存器。当它们的使能端信号有效且触发端信号有效时，输入 1D～8D 端的数据被锁存到 8D 触发器中并形成输出 1Q～8Q。74LS273、74LS377 的引脚封装如图 5.17 所示。

74LS273 的 1 脚为清除端，当该引脚为低电平时，1Q～8Q 输出被清 0；11 脚为锁存脉冲输入端。74LS377 所不同的是 1 脚为使能端，而没有清除端。

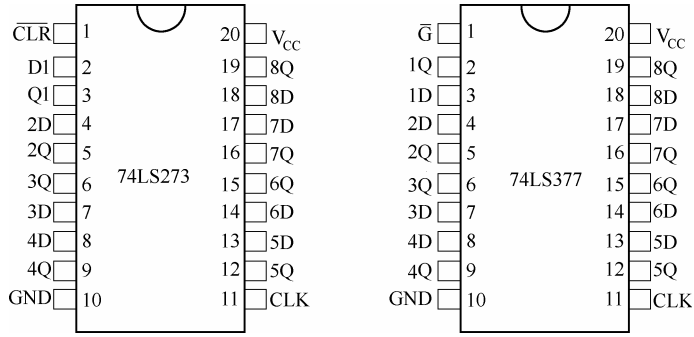


图 5.17　74LS273 和 74LS377 的引脚封装图

如图 5.18 所示的 74LS273 与 MCS-51 单片机连接电路，P2.7 与 $\overline{WR}$ 相或后作为 74LS273 的锁存信号，由于不要清 0，将 74LS273 清除端接高电平，74LS273 的端口地址为 7FFFH，将累加器 A 中的数据通过 74LS273 输出给外设的程序如下：

```
LS273：  MOV   DPTR，#7FFFH  ；将 74LS273 的端口地址送入 DPTR
         MOVX  @DPTR，A      ；将 A 中的数据送到 74LS273 进行输出
```

P2 口可以进行位操作，且 74LS273 只与 P2.7 有关，输出操作也可以通过如下程序实现：

```
LS273：  CLR   P2.7
```

```
MOVX  @R0，A
SETB   P2.7
```

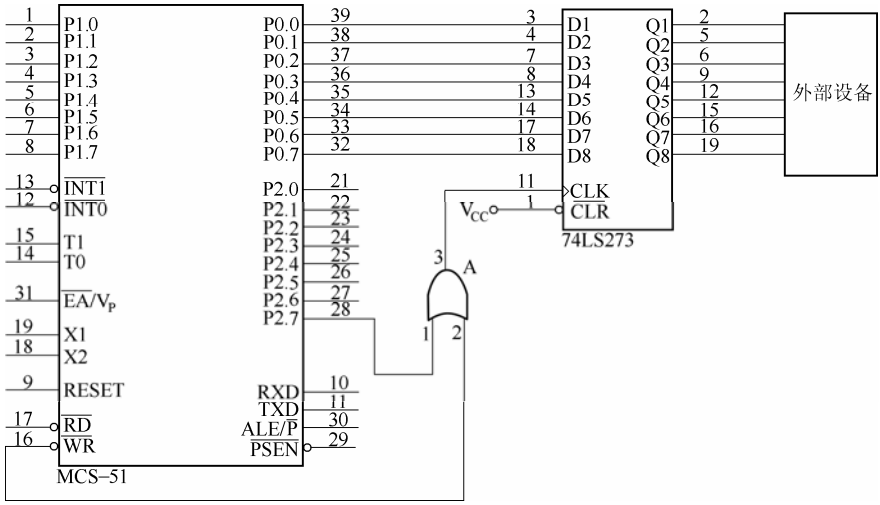


图 5.18　MCS-51 单片机与 74LS273 的接口电路图

5.3.2　可编程芯片 8155 的扩展

一些复杂的单片机应用系统，仅靠几片 74LS273 和 244 之类的简单扩展芯片已不能满足要求。必须扩展功能强的可编程接口芯片。下面通过 8155 芯片说明可编程并行接口芯片的扩展。

8155 是一个具有 RAM、I/O 口和计数器的通用可编程并行接口芯片。其内部资源主要有：

256 字节的静态 RAM（地址为××00H～××FFH）

两个可编程的 8 位并行 I/O 口 PA、PB

一个可编程的 6 位并行 I/O 口 PC

一个可编程的 14 位减法计数器 TC

具有 8 位地址锁存器

1. 8155 的结构

图 5.19 给出了 8155 的外部引脚分布示意图和内部逻辑框图。

8155 各引脚的功能及含义如下：

AD0～AD7：三态地址/数据复用总线，用于单片机和 8155 之间传送地址和数据。

IO/$\overline{M}$：IO 口/RAM 选择输入信号线，当为高电平时选择 I/O 口，为低电平时选择 RAM。

$\overline{CE}$：8155 的片选信号输入线，低电平有效。

ALE：地址允许锁存输入信号线，ALE 信号发生负跳变时把总线 AD0～AD7 的信息以及$\overline{CE}$、IO/$\overline{M}$的状态锁入片内锁存器。

$\overline{RD}$：读选通输入信号线，低电平有效。

$\overline{WR}$：写选通输入信号线，低电平有效。

TMRIN（TI）：计数器的外部计数脉冲输入端。

TMROUT（TO）：计数器的输出信号端，选择不同的工作模式，可输出方波或脉冲。

RESET：复位控制输入信号端，高电平有效，8155 复位后，A、B、C 口均置为输入方式。

PA0～PA7：A 口的 8 位并行 I/O 口线。

PB0～PB7：B 口的 8 位并行 I/O 口线。

PC0～PC5：C 口的 6 位并行 I/O 口线。

V_{CC}：电源输入端，单一+5V。

GND：接地端。

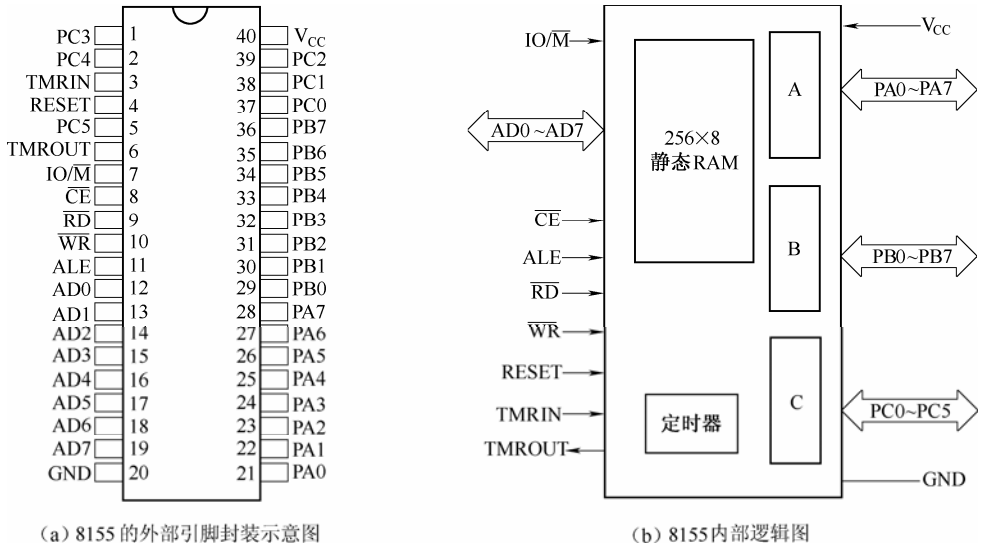


图 5.19　8155 的结构示意图

2. 8155 的RAM和I/O口地址编码

8155 在单片机应用系统中是按外部数据存储器统一编址的，为 16 位编址，高 8 位地址由 P2 口提供，经过译码器或直接控制 $\overline{CE}$ 和 IO/$\overline{M}$；而低 8 位地址由片内地址锁存器提供。当 IO/$\overline{M}$ 为低电平时，单片机对 8155 的 RAM 空间进行读/写操作，RAM 单元的低 8 位地址为 00H～0FFH；当 IO/$\overline{M}$ 为高电平时，单片机对 8155 的 I/O 口进行读/写操作。单片机对 8155 的内部寄存器和 RAM 的操作控制见表 5.11，8155 内部寄存器地址分配见表 5.12。

表 5.11　单片机对 8155 的操作控制表

控制信号				操　作
$\overline{CE}$	IO/$\overline{M}$	$\overline{RD}$	$\overline{WR}$	
0	0	0	1	读 RAM 单元（地址为××00H～××FFH）
0	0	1	0	写 RAM 单元（地址为××00H～××FFH）
0	1	0	1	读内部寄存器
0	1	1	0	写内部寄存器
1	×	×	×	无操作

3. 8155 的命令字和状态字

8155 提供的 PA 口、PB 口、PC 口以及定时器/计数器均可以编程控制，单片机可以通过写命令字来控制它们的操作形式，通过读状态字来判别它们的状态。命令字和状态字寄存器

共用一个口地址，命令字寄存器只能写而不能读，而状态字寄存器只能读不能写。

表 5.12　8155 内部寄存器地址分配表

低 8 位地址	I/O 口
××××× 000B	命令/状态寄存器（命令/状态口）
××××× 001B	PA 口
××××× 010B	PB 口
××××× 011B	PC 口
××××× 100B	定时器/计数器低字节寄存器
××××× 101B	定时器/计数器高字节寄存器

（1）8155 的命令字格式如下：

TM2	TM1	IEB	IEA	PC2	PC1	PB	PA

PA：PA 口输入/输出方式选择位。当该位为 0 时，PA 口为输入口；当该位为 1 时，PA 口为输出口。

PB：PB 口输入/输出方式选择位。功能与 PA 相同。

PC2 PC1：PA 口、PB 口、PC 口工作方式选择位。当 PC2 PC1=00 时，PA、PB 口为基本 I/O 口，PC 为输入口；当 PC2 PC1=01 时，PA、PB 为基本 I/O 口，PC 为输出口；当 PC2 PC1=10 时，PA 口为选通 I/O 口，PB 口为基本 I/O 口，PC5、PC4、PC3 为输出线，PC0 为 INTRA，PC1 为 BFA，PC2 为 $\overline{\text{STBA}}$；当 PC2 PC1=10 时，PA、PB 口均为选通 I/O 口，PC0 为 INTRA，PC1 为 BFA，PC2 为 $\overline{\text{STBA}}$，PC3 为 INTRB，PC4 为 BFB，PC5 为 $\overline{\text{STBB}}$。见表 5.13。

表 5.13　8155PA、PB、PC 的工作方式

端口	方式（PC2、PC1）			
	00B	01B	10B	11B
PA	基本输入/输出	基本输入/输出	选通输入/输出	选通输入/输出
PB	基本输入/输出	基本输入/输出	基本输入/输出	选通输入/输出
PC0	输入	输出	INTRA	INTRA
PC1	输入	输出	BFA	BFA
PC2	输入	输出	$\overline{\text{STBA}}$	$\overline{\text{STBA}}$
PC3	输入	输出	输出	INTRB
PC4	输入	输出	输出	BFB
PC5	输入	输出	输出	$\overline{\text{STBB}}$

IEA：PA 口中断请求使能位。当该位为 0 时，禁止 PA 口中断请求；当该位为 1 时，允许 PA 口中断请求。

IEB：PB 口中断请求使能位。功能与 IEA 相同。

TM2 TM1：定时器/计数器控制位。当 TM2 TM1=00 时，不影响定时器/计数器的工作，即空操作；当 TM2 TM1=01 时，若定时器/计数器正在工作立即停止定时器/计数器计数工作，若定时器/计数器没有工作则无操作；当 TM2 TM1=10 时，若定时器/计数器正在工作则待定时器/计数器发生计数溢出时停止计数工作，若定时器/计数器没有工作则无操作；当 TM2

TM1=11 时，若定时器/计数器没有工作则置方式和计数初值后立即启动定时器/计数器计数工作，若定时器/计数器正在运行则表示置新的方式和计数初值，计数结束后立即按新的方式和初值启动计数工作。

比如要使 PA 口、PB 口为基本输出口，PC 口为输出口，禁止 PA、PB 口中断，定时器无操作，则相应的命令字应该为：00001111H，即 0FH。

又如要使 PA 口为选通输入方式，PB 口为基本输出方式，允许 PA 口中断，启动定时器/计数器计数工作，则相应的命令字应该为：11010110B，即 0D6H。

（2）8155 的状态字格式如下：

—	TIMER	INTEB	BFB	INTRB	INTEA	BFA	INTRA

INTRA：PA 口的中断请求标志。该位为 0 表示 PA 口未产生中断请求；该位为 1 表示 PA 口产生了中断请求。

BFA：PA 口缓冲器满/空标志。当 PA 口作为输入口时，该位为 1 表示 PA 口缓冲器满；当 PA 口作为输出口时，该位为 0 表示 PA 口缓冲空。

INTEA：PA 口中断允许标志。该位为 1 表示 PA 口中断允许，该位为 0 表示 PA 口中断禁止。

INTRB：PB 口的中断请求标志。功能与 INTRA 相同。

BFB：PB 口缓冲器满/空标志。功能与 BFA 相同。

INTEB：PB 口中断允许标志。功能与 INTEA 相同。

TIMER：定时器/计数器中断请求标志。当定时器/计数器发生计数溢出时该位自动置 1，并由读状态字寄存器的操作或开始新的计数过程操作使其复位为 0。

（3）8155I/O 口的工作方式。

① 基本 I/O 方式：联络线由程序任意指定；对数据输入输出不起控制作用；没有中断能力；输出联络线完全由软件控制。

② 选通 I/O 方式：联络线由硬件固定确定；输入联络线可以起选通数据锁存作用；中断允许时输入联络线发生变化会形成中断请求信号；输出联络线受外设共同控制，不能随意输出。各联络信号的功能如下：

INTRA：PA 口以选通输入/输出方式工作时的中断请求输出信号。当 PA 口以选通方式每输入一个字节数据自动形成有效的高电平中断请求输出信号，PA 口一旦定义为选通输入方式其预置状态为低电平；当 PA 口以选通方式每输出一个字节数据自动形成有效的低电平中断请求输出信号，PA 口一旦定义为选通输出方式其预置状态为高电平。

BFA：PA 口缓冲器满/空输出信号。当 PA 口作为输入口时，该位为 1 表示 PA 口缓冲器满，PA 口一旦定义为选通输入方式其预置状态为 0；当 PA 口作为输出口时，该位为 0 表示 PA 口缓冲器空，PA 口一旦定义为选通输出方式其预置状态为 1。

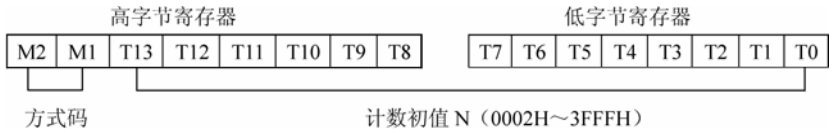
$\overline{\text{STBA}}$：数据选通输入信号，低电平有效。

INTRB、BFB 和 $\overline{\text{STBB}}$ 的功能与 INTRA、BFA 和 $\overline{\text{STBA}}$ 的相同。

4. 8155 的定时器/计数器

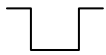
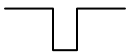
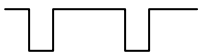
8155 片内的定时器/计数器是一个 14 位的减法计数器，它对 TIMERIN 端的时钟脉冲进行计数，并在达到最后计数值时给出一个方波或脉冲。计数器分为高 6 位和低 8 位计数长度

寄存器，初值由程序预置，14、15 位规定了定时器/计数器的 4 种输出方式，其格式如下：



4 种操作方式的选择及相应的输出波形如表 5.14 所示。

表 5.14　8155 定时器/计数器的四种操作方式及输出波形形式

M2M1	方　式	TIMEROUT 输出波形	说　明
0　0	单个方波		宽为 *N*/2（*N* 为偶数）或（*N*–1）/2（*N* 为奇数）个 TIMERIN 时钟周期
0　1	连续方波		低电平宽为 *N*/2（*N* 为偶数）或（*N*–1）/2（*N* 为奇数）个 TIMERIN 时钟周期；高电平宽为 *N*/2（*N* 为偶数）或（*N*+1）/2（*N* 为奇数）个 TIMERIN 时钟周期，自动恢复初值
1　0	单个脉冲		计数溢出时输出一个宽为一个 TIMERIN 时钟周期的负脉冲
1　1	连续脉冲		每次计数溢出时输出一个宽为 TIMERIN 时钟周期的负脉冲并自动重恢复初值

使用 8155 的定时器/计数器时，应先对它的高、低字节寄存器编程，设置操作方式和计数初值 N。然后对命令寄存器编程（命令字最高两位为 1），启动定时器/计数器计数。值得注意的是：硬件复位并不能初始化定时器/计数器为某种操作方式或者启动计数。定时器/计数器正在工作时也可将新的计数初值和工作方式打入计数长度寄存器，但在定时器/计数器使用新的计数值和方式之前，必须发一条启动命令，即使只希望改变计数值而不改变方式也必须如此。且写入的计数初值若为奇数，方波输出是不对称的。

若要停止定时器/计数器计数，需通过对命令寄存器编程（最高两位为 01 或者 10），使定时器/计数器立即停止计数或待定时器/计数器溢出时停止计数。硬件复位停止定时器/计数器计数。

8155 在计数过程中，由于是减法计数器，定时器/计数器的值并不直接代表从 TIMERIN 脚输入的时钟个数，必须通过下列步骤来获取 TIMERIN 引脚上输入的时钟数。

（1）停止计数。

（2）分别读取定时器/计数器的高、低字节寄存器并取其低 14 位数据。

（3）若这 14 位值为偶数，则当前计数状态等于此偶数右移一位的值；若为奇数，则当前计数状态等于此奇数右移一位后加上计数初值的一半的整数部分。

（4）当前计数值与初值之差即为 TIMERIN 引脚输入的时钟个数。

5. MCS-51 单片机与 8155 的接口

MCS-51 单片机与 8155 可以直接连接，如图 5.20 所示。在图中，将 P0.0～P0.7 与 8155 的 AD0～AD7 依次连接，用 P2.7 作为 8155 的片选信号，用 P2.0 作为 8155 的 RAM 或 I/O 端口选择操作信号，采用系统复位电路使 MCS-51 单片机与 8155 同时复位，其他的控制信号对应连接起来。

由图 5.20 可知，8155 的 RAM 单元地址空间为 7E00H～7EFFH，8155 的命令/状态寄存器的访问地址为 7F00H，PA 口、PB 口、PC 口的访问地址依次为 7F01H、7F02H、7F03H，

8155 的定时器/计数器的高、低字节寄存器的访问地址依次为 7F05H、7F04H。

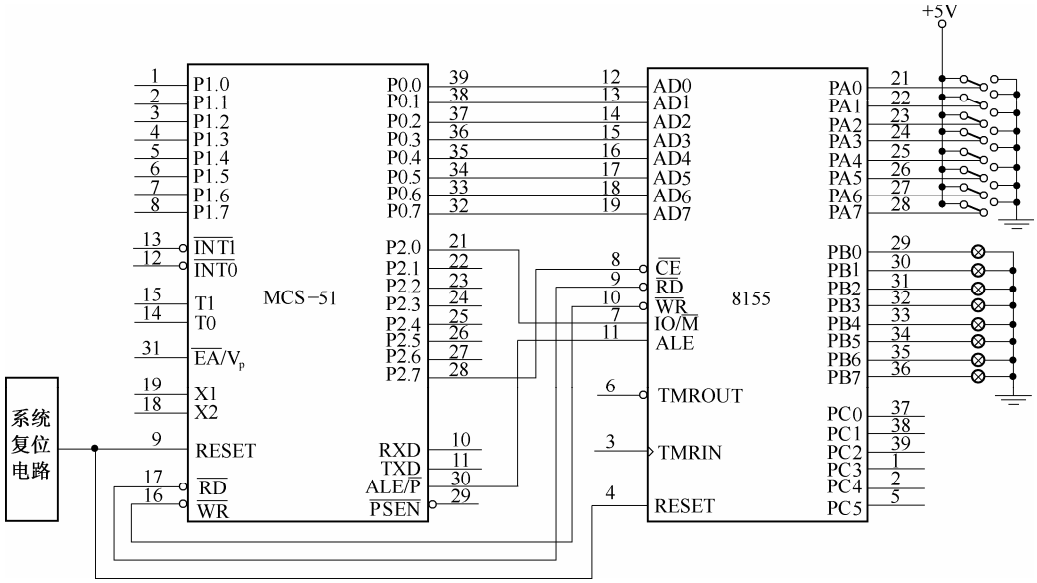


图 5.20　MCS-51 与 8155 的连接电路

（1）对 8155 的 RAM 单元的访问。假定在 MCS-51 单片机的内部 RAM 中，从 30H 开始放有一组数据，字节数 N，要求将该组数据存入 8155 RAM 中，起始地址为 7E00H，程序如下：

```
RAMWR:  MOV   R0，#30H
        MOV   DPTR，#7E00H
        MOV   R2，#N
RAMW:   MOV   A，@R0
        MOVX  @DPTR，A
        INC   R0
        INC   DPTR
        DJNZ  R2，RAMW
```

同样的道理，也可以将 8155 中的数据送入单片机中存储单元，请读者思考。

（2）对 8155 的 I/O 口的操作。如图 5.20 所示，假定在 8155 的 PA 口接了 8 个乒乓开关（开关的一个位置状态为高电平，开关的另一个位置状态为低电平），在 PB 口接了 8 个指示灯，要求 PB 口的 8 个指示灯按 PA 口的开关状态进行变化。

根据要求，8155 的 PA 口应定义为基本 I/O 输入口，PB 口应定义为基本 I/O 输出口，PC 口可以任意，所以 8155 的命令字为 00000010B，即 02H，程序如下：

```
        ORG   0030H
        MOV   DPTR，#7F00H  ；命令字/状态字寄存器地址送 DPTR
        MOV   A，#02H       ；设置命令字
        MOVX  @DPTR，A      ；送命令字，设置 PA、PB 口的工作方式
LOOP:   MOV   DPTR，#7F01H  ；PA 口地址送 DPTR
        MOVX  A，@DPTR      ；从 PA 口读取开关状态值
        INC   DPTR          ；指向 PB 口
```

```
        MOVX  @DPTR，A      ；将开关状态送到 PB 口指示
        SJMP  LOOP
        END
```

8155PA 和 PB 口工作在选通方式，可以采用查询，也可以采用中断等方式进行输入或输出，由于操作相对复杂，请读者思考并参阅相关书籍。

（3）对 8155 的定时器/计数器的操作。假定利用 8155 的定时器/计数器形成一个分频电路，要求将 8155 的 TIMERIN 引脚输入的脉冲信号进行 24 分频后经 TIMEROUT 引脚输出（TIMERIN 引脚每输入 12 个脉冲，TIMEROUT 引脚输出的状态发生一次翻转），参考图 5.20 所示电路。

根据要求，8155 的命令字为 11000010B，为实现 24 分频，则定时器低 8 位的值为 18H，以连续方波形式输出，则定时器高 8 位的值为 40H，程序如下：

```
        ORG   0030H
TIM:    MOV   DPTR，#7F04H ；定时器/计数器低字节寄存器地址送 DPTR
        MOV   A，#18H      ；分频数送定时器
        MOVX  @DPTR，A
        INC   DPTR         ；DPTR 指向定时器/计数器的高字节寄存器
        MOV   A，#40H      ；使定时器/计数器以连续方波输出
        MOVX  @DPTR，A
        MOV   DPTR，#7F00H ；将命令/状态字寄存器地址送 DPTR
        MOV   A，#0C2H     ；送命令字并启动定时器/计数器工作
        MOVX  @DPTR，A
        SJMP  $
        END
```

5.3.3 可编程芯片 8255A的扩展

8255A 是 Intel 公司生产的一种通用可编程接口电路，它的基本特性是：

（1）具有 3 个可编程的 8 位并行 I/O 口 PA、PB 和 PC。

（2）PA 具有基本 I/O、选通 I/O 和双向工作方式。

（3）PB 具有基本 I/O 和选通 I/O 工作方式。

（4）PC 口可以按位进行操作。

1. 8255A的结构

8255A 的内部机构图如图 5.21 所示。它由以下几个部分组成：

（1）外设接口部分。8255A 有 3 个 8 位并行 I/O 端口 PA、PB 和 PC，它们都可以作为输入口或输出口，但在结构和功能上有所不同。

PA 含有一个 8 位数据输出锁存/缓冲器和一个 8 位数据输入锁存器。

PB 含有一个 8 位数据输出锁存/缓冲器和一个 8 位数据输入锁存/缓冲器。

PC 含有一个 8 位数据输出锁存/缓冲器和一个 8 位数据输入缓冲器（不锁存）。

当数据传送不需要联络信号时，这 3 个端口都可以作为输入口或输出口。当 PA、PB 进行输入或输出需要联络信号时，PC 为 PA 和 PB 提供状态和控制信息。

（2）内部控制逻辑。内部控制逻辑主要由 A 组和 B 组控制电路组成。这两组电路根据 CPU 的命令字控制 8255A 的工作方式，每组控制电路从读/写控制逻辑接受各种命令，从内部数据总线接受控制字并发出适当的命令到各自相应的通道。它也可以根据 CPU 的命令字对 PC 的每一位实现按位“置位”或“复位”控制。A 组控制电路控制 PA 和 PC 的上半部分，B 组控制电路控制 PB 和 PC 的下半部分。

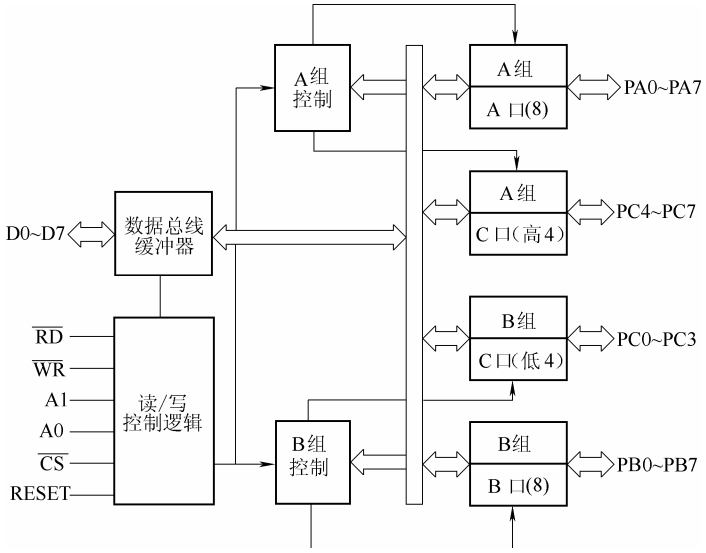


图 5.21　8255A 的内部结构框图

（3）CPU 接口部分。这部分电路主要由数据总线缓冲器和读/写控制逻辑组成。数据总线缓冲器是一个 8 位双向三态缓冲器，它是 8255A 与 CPU 数据总线的接口。所有数据的输入和输出，以及 CPU 用输出指令向 8255A 发出的控制字和用输入指令从 8255A 读入的外设状态信息，都是通过这个缓冲器传送的。

读/写控制逻辑的作用是从 CPU 的地址和控制总线上接受有关信号，转变成各种控制命令送到数据缓冲器以及 A 组、B 组控制电路，从而管理三个端口、控制寄存器与数据总线之间的传送操作。

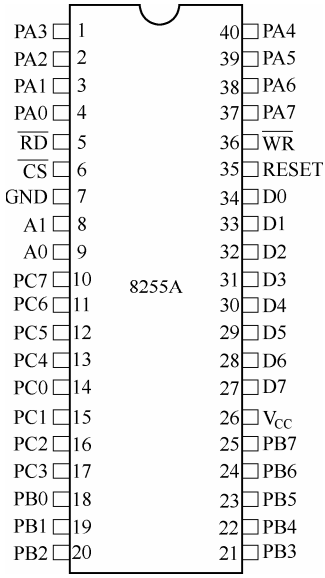


图 5.22　8255A 的引脚图

2. 8255A的引脚功能

8255A 采用 40DIP 封装，如图 5.22 所示，引脚功能如下：

D0～D7：数据总线，三态双向。

RESET：复位控制输入信号端，高电平有效。

$\overline{CS}$：片选信号输入线，低电平有效。

$\overline{RD}$：读选通输入信号线，低电平有效。

$\overline{WR}$：写选通输入信号线，低电平有效。

A1、A0：端口选择信号线，通过它们选择 PA、PB、PC 和命令寄存器，与 $\overline{CS}$、$\overline{RD}$、$\overline{WR}$ 信号一起，确定对 8255A 的操作状态，如表 5.15 所示。

PA0～PA7：PA 的 8 位 I/O 口线。

PB0～PB7：PD 的 8 位 I/O 口线。

PC0～PC7：PC 的 8 位 I/O 口线。

V_{CC}：电源输入端，单一+5V。

GND：接地端。

表 5.15　8255A 的端口选择和基本操作

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	操作
0	0	0	1	0	从 PA 输入
0	1	0	1	0	从 PB 输入
1	0	0	1	0	从 PC 输入
0	0	1	0	0	从 PA 输出
0	1	1	0	0	从 PB 输出
1	0	1	0	0	从 PC 输出
1	1	1	0	0	从命令口输出
×	×	×	×	1	数据总线为高阻态
1	1	0	1	0	非法状态
×	×	1	1	0	数据总线为高阻态

3. 8255A与单片机连接

如图 5.23 所示，8255A 与 MCS-51 单片机连接也比较方便，但由于 8255A 内部没有地址锁存器，因此需要外接 74LS373 地址锁存器，从这一点来说，它不如 8155 方便。根据连线图，8255A 的 $\overline{WR}$ 、$\overline{RD}$ 信号直接与单片机的 $\overline{WR}$ 、$\overline{RD}$ 相连，芯片选择信号 $\overline{CS}$ 通过单片机 P2.7 线选，地址锁存器锁存的最低两位地址与 8255A 的 A1、A0 连接以选择端口，假设未使用地址线取 1 状态，则 8255A 的端口地址如下：

PA　　7FFCH

PB　　7FFDH

PC　　7FFEH

命令口　　7FFFH

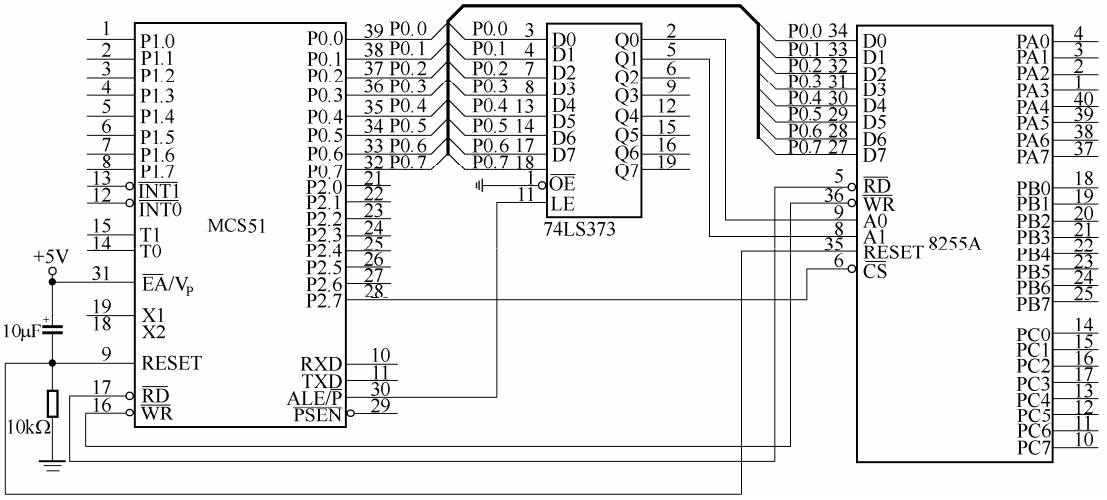


图 5.23　8255A 与 MCS-51 单片机的连接

4. 8255A的控制字

8255A 有两个控制字：方式选择控制字和 PC 按位置位/复位控制字。

（1）方式选择控制字。方式选择控制字的格式如下：

D7	D6	D5	D4	D3	D2	D1	D0

D7：D7 恒为 1，方式选择控制字的标志位。

D6D5：A 组方式选择，D6D5=00，为方式 0，D6D5=01，为方式 1，D6D5=1×，为方式 2。

D4：PA 的 I/O 选择，D4=1，PA 为输入口，D4=0，PA 口为输出口。

D3：PC4～PC7 的 I/O 选择，D3=1，PC4～PC7 为输入，D3=0，PC4～PC7 口为输出。

D2：B 组方式选择，D2=0，为方式 0，D2=1，为方式 1。

D1：PB 的 I/O 选择，D1=1，PB 为输入口，D1=0，PB 口为输出口。

D0：PC0～PC3 的 I/O 选择，D0=1，PC0～PC3 为输入，D0=0，PC0～PC3 口为输出。

8255A 的 PA 和 PB 在设定工作方式时，必须以 8 位为一个整体进行，而 PC 可以分为高 4 位和低 4 位分别选择不同的工作方式，这样 4 个部分可以按规定互相组合起来，十分灵活方便。

例如，假设 8255A 的 PA 工作于方式 0 输入，PB 和 PC 工作于方式 0 输出，则命令字为 10010000B=90H，以图 5.23 所示电路为例，则初始化程序为

```
MOV     DPTR，#7FFFH ；指针指向命令口
MOV     A，#90H      ；方式命令字
MOVX    @DPTR，A     ；写入 8255A 的命令寄存器
```

（2）按位置位/复位控制字。8255A 的 PC 可以按位进行操作，操作的命令字格式如下：

D7	D6	D5	D4	D3	D2	D1	D0

D7：恒为 0，按位置位/复位控制字的标志位。

D6～D4：没有使用，可以任意取值。

D3～D1：操作位的选择，D3D2D1=000 选择 PC0，D3D2D1=001 选择 PC1，……，D3D2D1=111 选择 PC7。

D0：置位/复位选择，D0=1 表示将选择的位置 1，D0=0 表示将选择的位清 0。

必须注意的是，虽然是对 PC 的某一位进行操作，但命令字必须从 8255A 的命令口写入。例如，如图 5.23 所示，要求从 PC7 输出方波，则程序如下：

```
        ORG     0030H
        MOV     DPTR，#7FFFH
LOOP：  MOV     A，#0FH      ；PC7=1 的命令字
        MOVX    @DPTR，A     ；写入 8255A 的命令寄存器
        MOV     R2，#80H     ；延时
        DJNZ    R2，$
        MOV     A，#0EH      ；PC7=0 的命令字
        MOVX    @DPTR，A     ；写入 8255A 的命令寄存器
        MOV     R2，#80H     ；延时
        DJNZ    R2，$
        LJMP    LOOP
        END
```

5. 8255A的工作方式

（1）工作方式 0。工作方式 0 为基本输入/输出方式。8255A 的 3 个端口都可以工作于这种方式，这种方式下 PA 和 PB 是 8 位整体选择为输入或输出，而 PC 的高低 4 位可分别选择其为输入或输出，作为输出口时，端口具有锁存功能，而作为输入口时，端口不具备锁存功能。

（2）工作方式 1。工作方式 1 为选通输入/输出方式。8255A 只有 PA 和 PB 有此工作方式，此时 PC 口的某些位将作为 PA 和 PB 的联络信号。输入和输出的情况有所不同。

① 方式 1 输入。8255A 工作于方式 1 输入情况下的功能如图 5.24 所示。

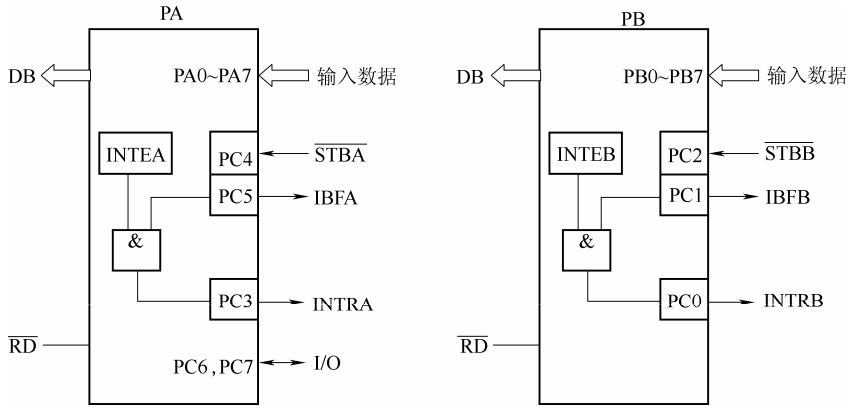


图 5.24　8255A 工作于方式 1 输入的功能

在方式 1 输入情况下，PC 的位被定义为：

PC4：PA 的选通信号 $\overline{\text{STBA}}$，低电平有效，有外设提供。当该信号有效时，8255A 的 PA 将外设提供的数据锁存。

PC5：PA 的输入缓冲器满信号 IBFA，高电平有效，由 8255A 输出给外设。当该信号有效时，表明外设送来的数据已到 PA 的输入缓冲器。该信号可以作为端口查询信号，只有当 PA 端口的数据被取走以后，该信号才变为低电平，端口才可以接收新的数据。

PC3：PA 的中断请求信号 INTRA，高电平有效，有 8255A 发出。在 INTEA=1 的条件下，当 $\overline{\text{STBA}}$ =1 和 IBFA=1 时，INTRA 被置为 1，当数据被取走后清除。

PC2：PB 的选通信号 $\overline{\text{STBB}}$，功能与作用同 $\overline{\text{STBA}}$。

PC1：PB 的输入缓冲器满信号 IBFB，功能与作用同 IBFB。

PC0：PB 的中断请求信号 INTRB，功能与作用同 INTRA。

PC6 和 PC7 可以作为自由的输入/输出线。

INTEA 和 INTEB 是 PA 和 PB 的中断允许位，它们是通过对 PC4 和 PC2 置 1 或清 0 来实现中断允许和中断禁止控制的，但这对 PC4 和 PC2 两引脚的功能（$\overline{\text{STBA}}$ 和 $\overline{\text{STBB}}$）并不影响。

② 方式 1 输出。8255A 工作于方式 1 输出情况下的功能如图 5.25 所示。在方式 1 输出情况下，PC 的位被定义为：

PC7：PA 的输入缓冲器满信号 $\overline{\text{OBFA}}$，低电平有效，由 8255A 输出给外设。当该信号有效时，表示 8255A 的 PA 中已有数据，外设可以将此数据取走。当 $\overline{\text{ACKA}}$ 到来时，该信号变为高电平。

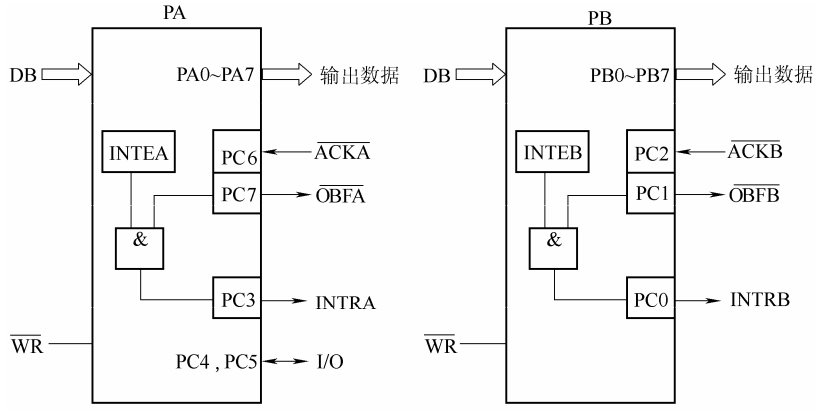


图 5.25　8255A 工作于方式 1 输出的功能

PC6：PA 的响应信号 $\overline{ACKA}$，低电平有效，当外设将数据取走后发回给 8255A 的应答信号。

PC3：PA 的中断请求信号 INTRA，高电平有效，有 8255A 发出。在 INTEA=1 的条件下，当 $\overline{OBFA}$ =1 和 $\overline{ACKA}$ =1 时，INTRA 被置为 1，$\overline{WR}$ 信号的上升沿使其复位。

PC1：PB 的选通信号 $\overline{OBFB}$，功能与作用同 $\overline{OBFA}$。

PC2：PB 的输入缓冲器满信号 $\overline{ACKB}$，功能与作用同 $\overline{ACKA}$。

PC0：PB 的中断请求信号 INTRB，功能与作用同 INTRA。

PC4 和 PC5 可以作为自由的输入/输出线。

INTEA 和 INTEB 是 PA 和 PB 的中断允许位，它们是通过对 PC6 和 PC2 置 1 或清 0 来实现中断允许和中断禁止控制的。

（3）工作方式 2。工作方式 2 为双向传送方式，8255A 只有 PA 有此工作方式。这种方式相当于 PA 方式 1 输入和输出的组合，其功能图如图 5.26 所示。

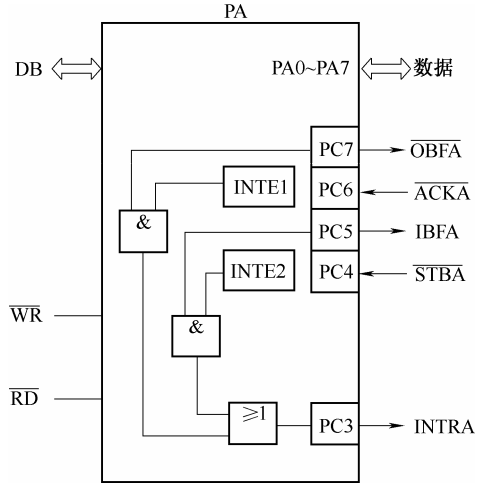


图 5.26　8255A 工作于方式 2 的功能

8255A 的 PA 工作于方式 2 时，PC3～PC7 作为其联络信号，与 PA 端口方式 1 输入和输出时相同，不再重复。若 PB 工作于方式 0，则 PC0～PC2 可作为自由的输入/输出线，但如果 PB 工作于方式 1，则 PC 的 8 条 I/O 线全部作为联络信号使用。

请读者思考：将如图 5.20 的 8155 换成 8255A，实现开关状态的输入与灯光控制，请画出电路，并编写相应的程序。

本 章 小 结

进行单片机功能扩展的前提是单片机已有的资源不能满足应用的要求。在选择单片机时，尽量选择具有足够程序存储器单元的单片机，因此程序存储器的扩展在现有的单片机应用中是极少的，数据存储器的扩展也是不多的。在单片机基本口不能满足要求时需要扩展 I/O 口，这种情况是比较常见的。本章要求：掌握 MCS-51 单片机的三总线形成；了解常用的程序存储器、数据存储器，以及它们与单片机如何进行连接，连接好的存储单元如何确定其地址；掌握用缓冲器、锁存器等扩展简单 I/O 接口电路；掌握 8155 可编程接口芯片的基本性能、端口与存储单元的编址方法，以及与单片机的连接电路，口的基本应用；了解 8155 定时器的应用；了解 8255A 的功能及应用。

习 题 5

5.1 如何形成 MCS-51 扩展系统所需的三总线？

5.2 在 MCS-51 单片机系统中，外部程序存储器和数据存储器共用 16 位地址线和 8 位数据，为什么不会发生冲突？

5.3 在一个 8032 扩展系统中，扩展了一片 DS1235、一片 8155、一片 74LS244，试画出系统的扩展电路图，并写出各器件的地址范围。

5.4 用单片机 P0 口扩展简单 I/O 口时，对输入接口和输出接口有什么基本要求？为什么？

5.5 在一个 89C51 系统中，扩展了一片 8155，试画出它们的接口连线图，要求写出将 89C51 片内 RAM 中的 20H～7FH 单元内容写入 8155RAM 的第一单元开始的空间中的程序。

5.6 在一个单片机系统中，扩展了一个 8155，要求 PA 口以选通的方式输出点亮 8 只走马灯，PB 口以周期 0.1s 输出点亮走马灯，试画出接口连线图，并写出相应的程序。设 f_{osc} 为 6MHz。

5.7 在一 f_{osc} 为 12MHZ 的 8031 系统中，扩展了一片 8155，用 8155 产生 1ms 的定时中断。试画出有关逻辑图，并编制 8031 的外部中断处理程序框图，使 8155 的 PA 口、PB 口的 16 位分别产生周期为 0.2s、0.4s、0.5s、1s、2s、4s、8s、16s、32s、64s、96s、128s、160s、192s、224s、256s 的方波信号。

第 6 章　MCS-51 系列单片机的接口技术

本章主要内容

键盘工作原理，键的去抖动。扫描工作原理及其在矩阵式键盘和动态数码显示中的应用，中断动态扫描显示。A/D 转换器的选型要点及常用 A/D 转换器与单片机的接口，编写实用的数据采集程序。D/A 转换器的选型要点及常用 D/A 转换器与单片机的接口，V/I 转换电路及应用。干湿接点的开关量输入技术，功率开关量输出技术。SPI 和 I^2C 串行总线接口技术。

6.1　键盘接口技术

在单片机应用系统中，键盘是人机交流的重要组成部分，用于向单片机应用系统输入数据或控制信息。键盘形式一般有独立式键盘和矩阵式键盘两种。独立式键盘的结构简单，但占用的资源多，通常用在按键数量较少的场合，大多数单片机应用采取这种方式；矩阵式键盘的结构相对复杂些，但占用的资源较少，通常用在按键数量多的场合。

6.1.1　键的特性

由于键的按下与释放是通过机械触点的闭合与断开来实现的，因机械触点的弹性作用，在闭合与断开的瞬间均有一个抖动过程，所以键闭合与断开会产生的电压波形如图 6.1 所示，抖动时间一般在 5ms～10ms。这个抖动对判断键是否按下或释放有重大影响，因此必须消除键的抖动，只有这样，才能可靠地判断键的状态。

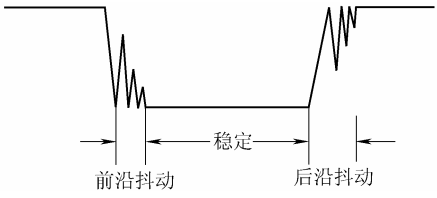
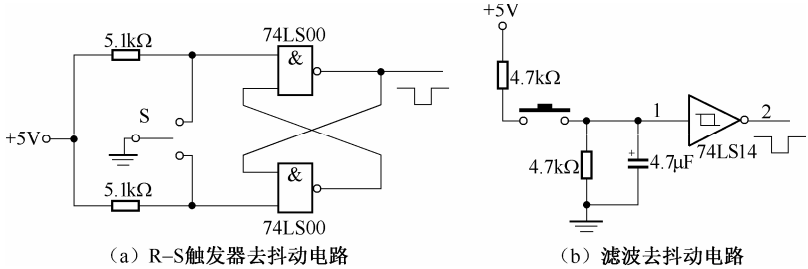


图 6.1　按键抖动信号波形

在单片机应用系统中，消除抖动有硬件和软件两种方法。硬件去抖动方法主要有利用 R-S 触发器和滤波电路，如图 6.2（a）和 6.2（b）所示。软件去抖动通常是程序检测到键被按下或释放时，延时 10ms 后再检测键是否仍然闭合或断开，若是则确认是一次真正的闭合或断开，否则就忽略此次按键或释放。



（a）R–S触发器去抖动电路　　（b）滤波去抖动电路

图 6.2　按键去抖动电路

6.1.2 独立键盘接口技术

1．独立键盘的结构

如图 6.3 所示，每一个按键的电路是独立的，占用一条数据线，当其中任意一按键按下时，它所对应的数据线的电平就变成低电平，读入单片机就是逻辑 0，表示键闭合，若无键闭合，则所有的数据线的电平都是高电平。

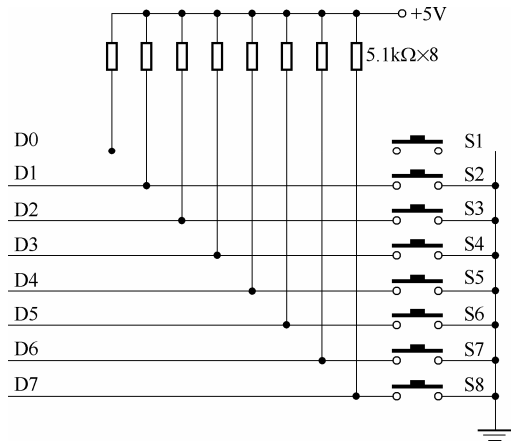


图 6.3　独立键盘结构

2．独立键盘与单片机接口

在单片机应用时，应遵循尽量不扩展的原则，因此，以单片机基本口为例来分析独立键盘的实现方法，如图 6.4 所示。

设 K1 闭合将 20H 单元内容加 1，K2 闭合将 20H 单元内容减 1，K3 闭合将 21H 单元清 0，K4 闭合将 21H 单元置 FFH，若同时有两个以上按键闭合，将不做任何操作。

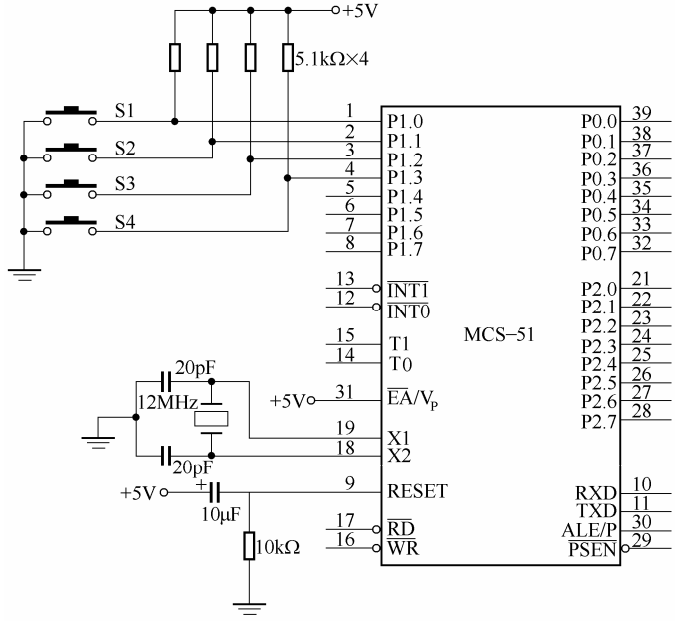


图 6.4　独立键盘与单片机接口

由于 P1 口是一个准双向口，因此首先置 P1 口为输入口。然后读取 P1 口的引脚状态，判断是否有键被按下，若有键被按下，则利用延时程序延时 10ms 时间再判断键是否依然被按下，以确保是一次真正的按键操作。在这里关键的问题是如何判断单键被按下还是有两个以上的键被按下，在此通过比较指令来判别，这是因为四个按键分别接在 P1.0、P1.1、P1.2 和 P1.3 引脚上，因此读入后经处理得到相应的单键被下代码分别是 01H、02H、04H 和 08H。程序流程图如图 6.5 所示。

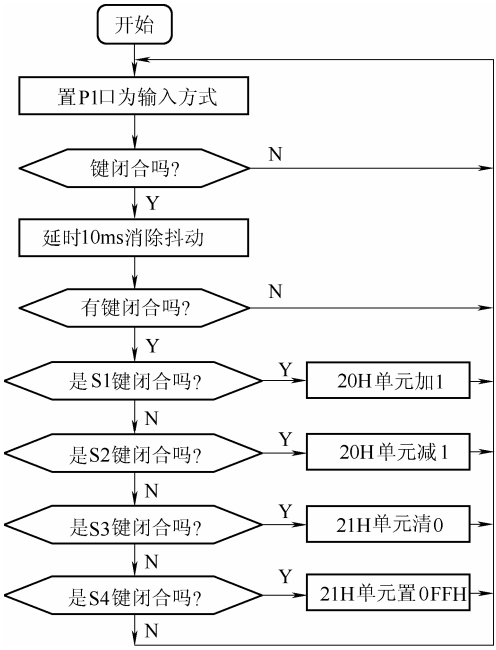


图 6.5　独立键盘程序框图流程图

程序如下：

```
        ORG    0030H
KB:     MOV    P1，#0FFH          ；置 P1 口为输入口
        MOV    A，P1              ；读键状态
        CPL    A
        ANL    A，#0FH            ；屏蔽高 4 位
        JZ     KB                 ；无键闭合则返回
        ACALL  D10MS              ；延时去抖动
        MOV    A，P1              ；再读键状态
        CPL    A
        ANL    A，#0FH
        JZ     KB                 ；无键闭合则返回
        CJNE   A，#01H，KB01
        INC    20H                ；K1 键闭合，20H 单元加 1
        SJMP   KB
KB01:   CJNE   A，#02H，KB02
        DEC    20H                ；K2 键闭合，20 单元减 1
```

```
        SJMP    KB
KB02:   CJNE    A，#04H，KB03
        MOV     21H，#00H          ；K3 键闭合，21H 单元清 0
        SJMP    KB
KB03:   CJNE    A，#08H，KB
        MOV     21H，#0FFH         ；K4 键闭合，21H 单元置 FF
        SJMP    KB                 ；若有两键以上闭合则返回
        END
```

为了节省篇幅，延时程序没在程序中给出。

上例是利用单片机 P1 口实现的，当然也可以通过单片机其他基本口和其他扩展口来实现，只需对程序中涉及口的地址做相应变化即可。

6.1.3 矩阵键盘接口技术

1．矩阵键盘的结构

在按键较多时，为了少占用单片机 I/O 线资源，通常采用矩阵式键盘，如图 6.6 所示，有 4 行 4 列构成了 16 个键阵。

每一行线与列线的交叉处是互不相通的，而是通过一个按键来接通。D0～D3 作为行线，D4～D7 作为列线。

在键盘处理程序中，首先确定是否有键按下，下一步再识别是哪一个键被按下，通常使用的一种方法是扫描法，逐行或逐列地进行。

下面以如图 6.6 所示 4×4 键盘为例，介绍矩阵键盘的工作原理。

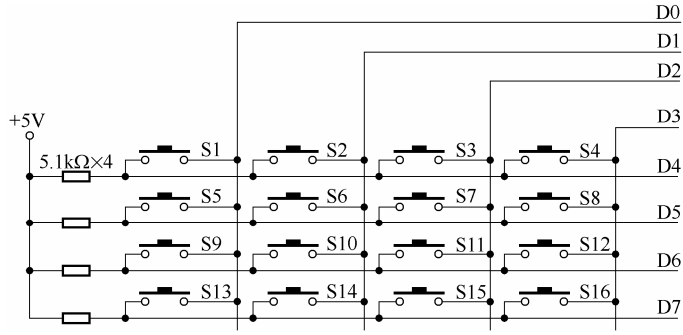


图 6.6　矩阵键盘结构

（1）使列线 D0～D3 都输出 0，检测行线 D4～D7 的电平。如果 D4～D7 上的电平全为高，则表示没有键被按下。如果 D4～D7 上的电平不全为高，则表示有键被按下。

（2）如果没有键闭合，就返回扫描。如果有键闭合，再进行逐列扫描，找出闭合键的键号。先使 D0=0，D1～D3=1，检测 D4～D7 上的电平，如果 D4=0，表示 K1 键被按下；同理，如果 D5～D7=0，分别表示 K5、K9、K13 键被按下；如果 D4～D7=1，则表示这一列没有键被按下。再使 D1=0，D0、D2、D3 为 1，对第二列进行扫描，这样依次进行下去，直到把闭合的键找到为止。

2．矩阵键盘工作方式

上面讲了矩阵键盘的工作原理，怎样实现键盘扫描呢？主要有程序扫描方式、定时扫描

方式和中断扫描方式三种。

（1）程序扫描方式：程序扫描方式是在 CPU 不执行别的程序时，对键盘进行扫描，占用 CPU 的时间比较多，当 CPU 执行其他功能程序时，就不再响应键盘的要求。

图 6.7 所示为利用单片机 P1 口实现的 4×4 矩阵键盘，行线为 P1.4～P1.7，列线为 P1.0～P1.3。程序流程图如图 6.8 所示。

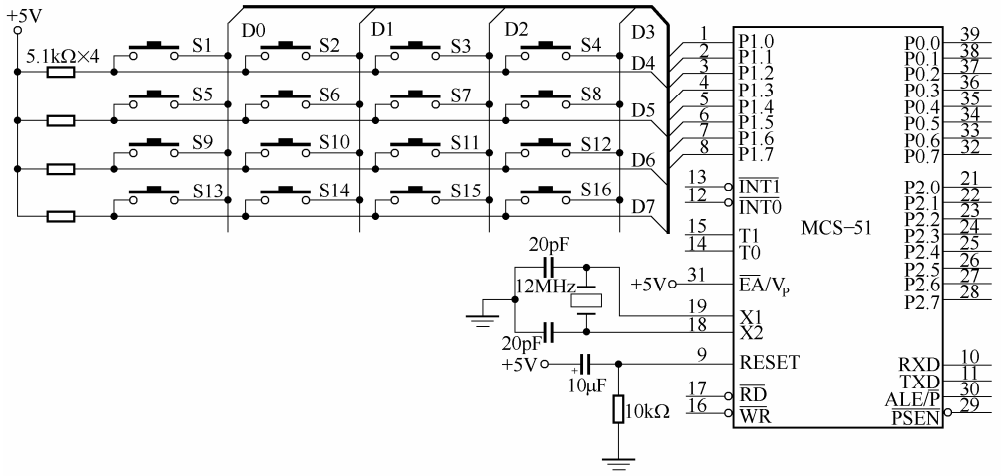


图 6.7　矩阵键盘与单片机接口

① 判别有无键按下。将 P1 口低 4 位输出为 0，把 P1 口高 4 位读入单片机，若 P1.4～P1.7 的状态全为 1，表明无键按下，否则，表明至少有一个键按下。

② 消除按键抖动的影响。在判断有键按下后，调用延时子程序，延时时间约 10ms，再判断 P1.4～P1.7 的状态，如果仍然是有键按下的状态，则确认键被真正按下，否则当作按键抖动处理。

③ 求按键的位置。首先使 P1.0 为低电平，读入 P1.4～P1.7 的状态，若 P1.4=0，表示第 1 行第 1 列的键（K1）按下，P1.5=0，表示 2 第 1 行第 1 列的键（K5）按下，依次类推，P1.6 和 P1.7 为 0 表示 K9 和 K13 闭合。

然后再使 P1.1 为低电平，扫描第 2 列，这样可以确定第 1、2、3、4 行第 2 列的按键状况。由此可见，列扫描号和行读入状态就可以确定按键的位置，我们把列扫描号称为列号，把行读入状态称为行号，列号+行号=键值，它代表了键的位置。

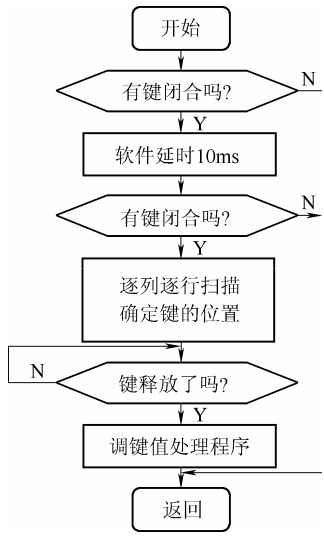


图 6.8　程序键盘扫描程序框图

④ 键闭合一次只进行一次处理，因此在键被释放之后才进行键的处理。

若将程序键盘扫描程序设计成子程序，则程序如下：

```
KSCAN:  ACALL  KEYS1          ；调用判键闭合子程序
        JNZ    KEY1           ；有键闭合则转至去抖动
        AJMP   RETURN         ；无键闭合则返回
KEY1:   ACALL  D10MS          ；调用 10ms 延时程序
        ACALL  KEYS1          ；再次调用判键闭合子程序
```

```
        JNZ     KEY2            ；确认有键闭合，开始扫描
        AJMP    RETURN          ；无键闭合则返回
KEY2：  MOV     R2，#0FEH       ；送首列扫描字
        MOV     R4，#00H        ；送首列号
KEY0：  MOV     A，R2
        MOV     P1，A
        MOV     A，P1           ；扫描字从 P1 口送出
        JB      ACC.4，LINE1    ；第 1 行无键闭合，转第 2 行
        MOV     A，#00H         ；第 1 行首键号送 A
        AJMP    KPV             ；转键值计算程序
LINE1： JB      ACC.5，LINE2    ；第 2 行无键闭合，转第 3 行
        MOV     A，#04H         ；第 2 行首键号送 A
        AJMP    KPV             ；转键值计算程序
LINE2： JB      ACC.6，LINE3    ；第 3 行无键闭合，转第 4 行
        MOV     A，#08H         ；第 3 行首键号送 A
        AJMP    KPV             ；转键值计算程序
LINE3： JB      ACC.7，NEXT     ；第 4 行无键闭合，转下 1 列
        MOV     A，#0CH         ；第 4 行首键号送 A
KPV：   ADD     A，R4           ；计算键值
        PUSH    ACC
KEY3：  ACALL   KEYS1           ；等待键释放
        JNZ     KEY3
        POP     ACC
        SETB    FLAG            ；置有键按下标志
        SJMP    KEY4
RETURN：CLR     FLAG            ；清有键按下标志
KEY4：  RET
NEXT：  INC     R4              ；列号加 1
        MOV     A，R2
        CJNE    A，#0F7H，RETURN；判断 4 行是否都已扫描完，完则返回
        RL      A               ；为扫描下 1 行做准备
        MOV     R2，A
        AJMP    KEY0            ；开始扫描下 1 列
KEYS1： MOV     P1，#0F0H       ；判键闭合子程序
        MOV     A，P1
        CPL     A
        ANL     A，#0F0H
        RET
D10MS： MOV     R7，#14H        ；10ms 延时子程序
DLY：   MOV     R6，#0F8H
DLY1：  DJNZ    R6，DLY1
```

```
        DJNZ    R7，DLY
        RET
```

注意，在主程序中应对 FLAG 标志位进行定义。

（2）定时扫描方式：程序判断扫描消耗大量 CPU 时间，降低了单片机的工作效率。由于机械按键被按下到释放有相当长一段时间，没有必要不间断地监视按键状态，可以间断扫描键盘，这个间断时间可以用中断来实现。定时扫描方式就是利用定时器产生定时中断，在中断服务时扫描键盘。这种方式能及时响应键输入，还能去掉软件去抖动时间。定时扫描方式的电路与程序扫描方式相同，其程序流程图如图 6.9 所示。

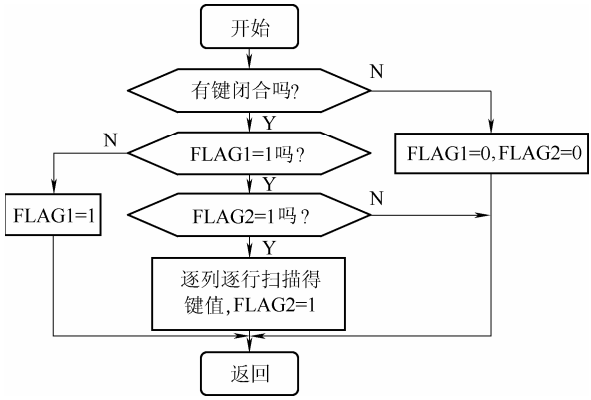


图 6.9　定时键盘扫描程序框图

定时扫描方式要求设置两个标志：去抖动标志 FLAG1 和处理标志 FLAG2。定时扫描时若无键闭合，将 FLAG1 和 FLAG2 标志清除后即返回。当有键闭合时，先查 FLAG1 标志，若 FLAG1=0，表示尚未作去抖动处理，将 FLAG1 置 1 后即返回。中断返回后要经 10ms 才再次中断，这个时间即为去抖动时间。若 FLAG1=1，说明已经作了去抖动处理，接着查 FLAG2 标志。若 FLAG2=0，说明还没有对该键做处理，因此进行键扫描，得到键值，以待进一步分析处理，并将 FLAG2 置 1 后返回。若 FLAG2=1，说明该键已经做过处理，避免重复处理，直接返回。

为了使程序完整，设系统时钟为 12MHz，将键值存放于 30H 单元，20H.0 和 20H.1 分别作为 FLAG1 和 FLAG2。程序如下：

```
FLAG1   BIT     20H.0           ；定义抖动标志
FLAG2   BIT     20H.1           ；定义处理标志
KEYV    EQU     30H             ；定义键值单元
        ORG     0000H
        AJMP    MAIN
        ORG     000BH
        AJMP    INTT0
MAIN：  MOV     TMOD，#01H      ；T0 为方式 1，定时 10ms
        MOV     TL0，#0F0H
        MOV     TH0，#0D8H
        MOV     IE，#82H
        SETB    TR0
```

```
HERE：  SJMP   HERE              ；主程序踏步
INTT0：  MOV    TL0，#0F0H       ；中断服务，重新给定时器赋初值
        MOV    TH0，#0D8H
        ACALL KEYS1              ；判断键是否按下
        JNZ    KB1               ；有键按下则转至 KB1
        CLR    FLAG1             ；清除抖动标志
        CLR    FLAG2             ；清除处理标志
        AJMP   KB2               ；中断返回
KB1：   JB     FLAG1，KB3        ；判抖动标志是否有效
        SETB   FLAG1             ；无效则设置成有效
        AJMP   KB2               ；返回
KB3：   JB     FLAG2，KB2
        MOV    R2，#0FEH         ；送首列扫描字
        MOV    R4，#00H          ；送首列号
KEY0：  MOV    A，R2
        MOV    P1，A
        MOV    A，P1             ；扫描字从 P1 口送出
        JB     ACC.4，LINE1      ；第 1 行无键闭合，转第 2 行
        MOV    A，#00H           ；第 1 行首键号送 A
        AJMP   KPV               ；转键值计算程序
LINE1：  JB     ACC.5，LINE2      ；第 2 行无键闭合，转第 3 行
        MOV    A，#04H           ；第 2 行首键号送 A
        AJMP   KPV               ；转键值计算程序
LINE2：  JB     ACC.6，LINE3      ；第 3 行无键闭合，转第 4 行
        MOV    A，#08H           ；第 3 行首键号送 A
        AJMP   KPV               ；转键值计算程序
LINE3：  JB     ACC.7，NEXT       ；第 4 行无键闭合，转下 1 列
        MOV    A，#0CH           ；第 4 行首键号送 A
KPV：   ADD    A，R4             ；计算键值
        MOV    KEYV，A           ；存放键值
        SETB   FLAG2             ；设置处理标志
        AJMP   KB2
NEXT：  INC    R4                ；列号加 1
        MOV    A，R2
        CJNE   A，#0F7H，KB2     ；判断 4 行是否都已扫描完，完则返回
        RL     A                 ；为扫描下 1 行做准备
        MOV    R2，A
        AJMP   KEY0              ；开始扫描下 1 列
KB2：   RETI
        END
```

前面讲的独立键盘同样可以采用定时扫描方式来实现，请读者思考如何进行。

（3）中断扫描方式：中断扫描方式只有在键被按下时，才进行键盘扫描，能在最快时间内对按键进行响应，占用 CPU 的时间最少，运行效率高。无论是独立键盘，还是矩阵键盘都可以采用中断扫描方式。限于篇幅，在此仅以独立键盘为例画出硬件原理图，如图 6.10 所示，未给出程序清单，请读者思考。

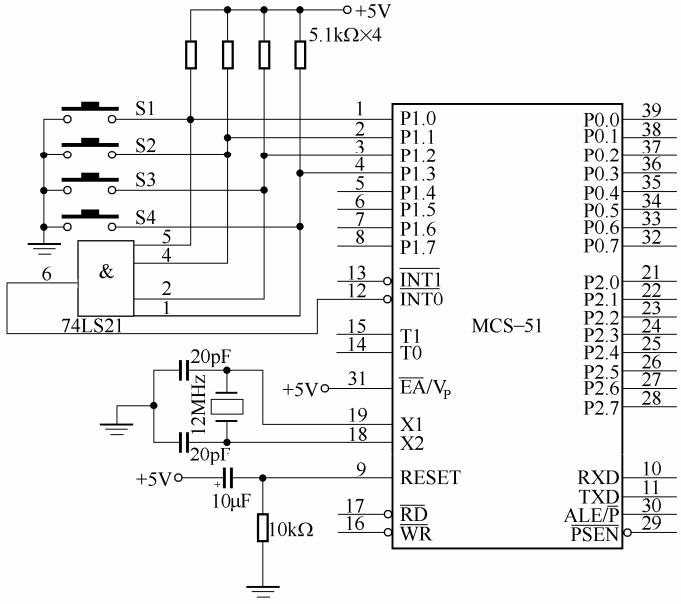


图 6.10　独立键盘以中断方式与单片机接口

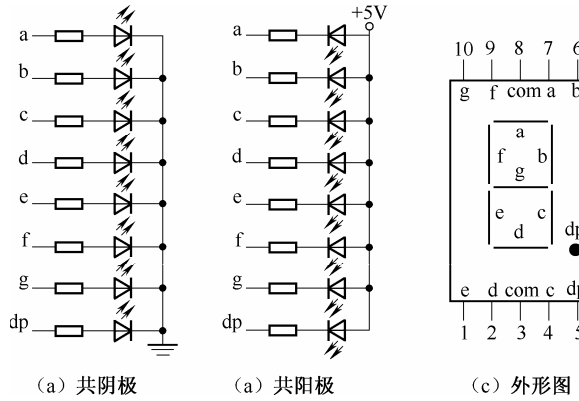
6.2　数码显示接口技术

同键盘一样，显示器也是人机交流的重要组成部分。计算机的运行结果和运行状态可以通过显示器显示出来。单片机应用系统中常用的显示器有 LED 和 LCD 两中方式，LED 数码显示最为普遍，但由于低功耗的要求，LCD 显示越来越被广泛地使用。在本节讨论 LED 数码显示技术，在下一节简要介绍 LCD 显示技术。

6.2.1　数码显示原理

1．LED显示器的结构与原理

LED 数码显示器是由若干个发光二极管组成的，当发光二极管导通时，相应的点或线段发光，将这些二极管排成一定图形，控制不同组合的二极管导通，就可以显示出不同的字形。单片机应用系统中常用的 LED 显示器为七段显示器，再加上有一个小数点，因此也可把它称为八段显示器。结构形式有共阴极和共阳极两种，它的结构图如图 6.11 所示。共阴极是把所有发光二极管的阴极连起来，通常接地，通过控制每一只发光二极管的阳极电平来使其发光或熄灭，阳极为高电平发光，为低电平熄灭。共阳极是把所有发光二极管的阳极连起来，通常接高电平（如+5V），通过控制每一只发光二极管的阴极电平来使其发光或熄灭，阴极为低电平发光，为高电平熄灭。图 6.11（c）当中的 com 端在应用时作为位选端，8 只发光二极管被分成两组，所以有两个 com 端，在使用时把它们并联起来。



（a）共阴极　　（a）共阳极　　（c）外形图

图 6.11　LED 显示器结构图

必须注意的是，在图中的电阻并非数码管内部就有的电阻，它们是需外接的限流电阻，如果不限流将造成发光二极管的烧毁！限流电阻的取值一般使流经发光二极管的电流在10mA～20mA，由于高亮度数码管的使用，电流还可以取得小一些。

2．数码管段选码

为了在 LED 显示器上显示某个字符，必须在它的 8 位段选线上加上相应的电平组合，即一个 8 位数据，这个数据就叫该字符的段选码。通常用的段选码的编码规则如下所示。

dp	g	f	e	d	c	b	a

忽略小数点的七段 LED 显示器的段选码如表 6.1 所示。

表 6.1　七段 LED 显示器段选码表

显示字符	共阴极段选码	共阳极段选码	显示字符	共阴极段选码	共阳极段选码
0	3FH	C0H	b	7CH	83H
1	06H	F9H	C	39H	C6H
2	5BH	A4H	d	5EH	A1H
3	4FH	B0H	E	79H	86H
4	66H	99H	F	71H	8EH
5	6DH	92H	P	73H	8CH
6	7DH	82H	U	3EH	C1H
7	07H	F8H	y	6EH	91H
8	7FH	80H	Γ	31H	CEH
9	6FH	90H	8.	FFH	00H
A	77H	88H	灭	00H	FFH

以上是不带小数点的字段选码，很容易得到带小数点的字段选码。

6.2.2　静态显示技术

静态显示就是当数码管显示某一字符时，相应的发光二极管连续恒定地处于点亮或熄灭状态，直到更换显示内容为止。采用这种显示方式占用的硬件资源多，以七段 LED 显示器为例，如果用软件进行字段译码，每显示一个字符就需要一个锁存器，如果用硬件进行字段译

码，每显示一个字符就需要一个锁存译码器。静态显示的由于数码管连续地工作，因此功耗大，但程序简单，亮度高。随着高亮度数码管的出现，动态显示同样可以达到很好的显示效果，所以在多数应用情况下，特别是显示位数比较多的情况下，不会采取静态显示方式，而采取动态显示方式。

下面举一个例子。如图 6.12 所示，单片机 P2 口通过 74LS245 驱动后接一只共阴极数码管，每一段都串有限流电阻，让其循环显示 0～9，每个数字停留显示的时间为 0.1s。

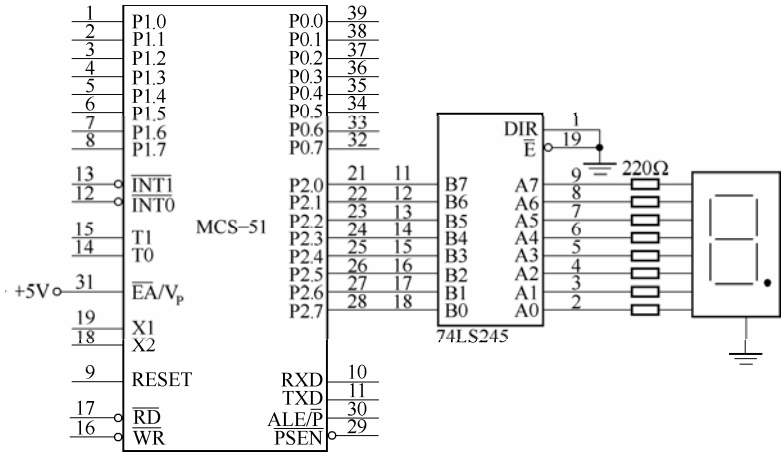


图 6.12　LED 静态显示电路

程序如下：

```
        ORG   0030H
DISP0:  MOV   DPTR，#SEG        ；字段码首地址
DISP1:  CLR   A                 ；从 0 开始显示
        MOV   R2，#0AH          ；显示计数器
DISP3:  MOVC  A，@A+DPTR        ；查字符段选码
        MOV   P2，A             ；从 P2 口输出显示
        MOV   R3，#0AH          ；停留 0.1s
DISP4:  ACALL D10MS
        DJNZ  R3，DISP4
        INC   A
        DJNZ  R2，DISP2
        AJMP  DISP1             ；又从 0 开始显示
D10MS:  MOV   R7，#14H          ；10ms 延时子程序
DLY:    MOV   R6，#0F8H
DLY1:   DJNZ  R6，DLY1
        DJNZ  R7，DLY
        RET
SEG:    DB    3FH，06H，5BH，4FH，66H
        DB    6DH，7DH，07H，7FH，6FH
        END
```

6.2.3 动态显示技术

在多位 LED 显示时，为了降低成本和功耗，将所有位的段选线并联起来，由一个 8 位口控制，由另一个端口进行显示位的控制。但是，由于段选线是公用的，要让各位数码管显示不同的字符，就必须采用扫描方式，即动态扫描显示方式。首先从段选线上送出字段码，再控制位选线，字符就显示在指定位置上，持续 1～5ms 时间，然后关闭所有显示；接下来又送出新的字段码，按照上述过程又显示在新的位置上，直到每一位数码管都扫描完为止，即为一个扫描周期。由于人的视觉停留效应，因此当扫描周期小到一定程度时，人就感觉不出字符的移动或闪烁，觉得每位数码管都一直在显示，达到一种稳定的视觉效果。

动态扫描显示的扫描方式有程序控制扫描和定时中断扫描两种。程序控制扫描方式要占用许多 CPU 时间，在计算机的任务较重时，难以得到很好的效果，所以在实际应用中常采用定时中断扫描方式，这种方式是每隔一定时间（如 1ms）去显示一位数码管，假设有 8 位数码管，显示扫描周期为 8ms，显示效果十分良好。

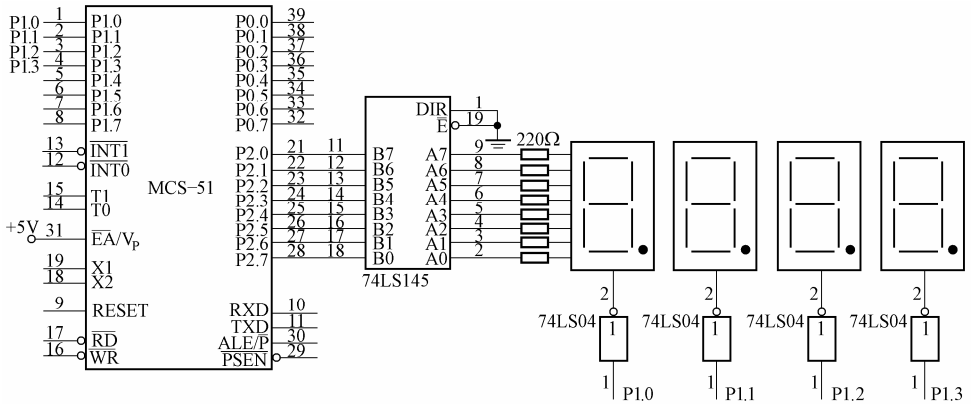


图 6.13　LED 动态显示电路

下面以定时中断扫描方式为例，如图 6.13 所示，在 4 位数码管上分别显示 1234。单片机定时器 T0 定时 1ms，要显示的 4 位数据 1、2、3、4 放到显示缓冲单元 30H～33H。程序如下：

```
        ORG     0000H
        AJMP    MAIN
        ORG     000BH
        AJMP    INTT0
MAIN:   MOV     TMOD，#01H          ；T0 定时 1ms 中断初始化
        MOV     TL0，#18H
        MOV     TH0，#0FCH
        MOV     IE，#82H
        SETB    TR0
AGAIN:  MOV     R0，#30H            ；显示缓冲区首地址
        MOV     R2，#01H            ；显示位控制字
NEXT:   MOV     A，R2
        JB      ACC.4，AGAIN        ；4 位扫描完又重复
```

```
        SJMP    NEXT                ；4 位未完等待显示下一位
INTT0：  MOV    TL0，#18H           ；重为定时器赋初值
        MOV     TH0，#0FCH
        MOV     P1，#0FFH           ；关所有显示
        MOV     A，@R0              ；取显示数字
        MOV     DPTR，#SEG
        MOVC    A，@A+DPTR          ；查字段码表的段选码
        MOV     P2，A               ；输出段选码
        MOV     A，R2
        MOV     P1，A               ；输出位控制字
        RL      A                   ；为显示下一位做准备
        MOV     R2，A
        INC     R0
        RETI
SEG：   DB      3FH，06H，5BH，4FH，66H
        DB      6DH，7DH，07H，7FH，6FH
        END
```

请读者按照上述例子试验一下，将定时扫描时间按 50ms，20ms，10ms，1ms 逐步减小，观察 LED 的显示效果，就可充分理解扫描显示技术。

6.3 液晶显示技术

6.3.1 液晶显示器简介

液晶显示器是一种功耗极低的显示器件。近年来，液晶显示技术发展十分迅猛，市面上出现了各种各样的 LCD 显示板。液晶显示器具有低压微功耗、平板型结构、显示信息量大、没有电磁辐射、寿命长等特点，现越来越广泛地用在各种智能仪器仪表中。目前市面上的液晶显示器主要有笔段型、点阵字符型和点阵图形型等三种形式。笔段型通常有 7 段、8 段、9 段、14 段、16 段等，主要用来显示数字、西文字母或某些字符，这与 LED 数码管相似；点阵字符型主要有 5×8、5×11 等点阵块组成，主要用来显示字符、数字、符号等；点阵图形型是在平面上排成多行多列的晶格阵列，可以显示图形和汉字等复杂的信息。

各种形式的液晶显示器都有与之配套的驱动芯片。笔段型驱动器如 HD44100H 等；点阵字符型驱动器如 HD44780 等；点阵图形型驱动器如 T6963C 等。

6.3.2 液晶显示器与单片机接口

笔段型液晶显示器与单片机接口同 LED 数码管与单片机接口类似，在此就不再讲了；而点阵图形型液晶显示器实现复杂图形的显示也比较麻烦，在此也不讲。在此以应用十分普遍的点阵字符型液晶显示器为例来讲解液晶显示器如何与单片机接口。

液晶显示器通常把驱动电路做在一块，形成液晶显示模块，用户不必了解驱动器与显示器如何连接，完全可以把液晶显示模块看作一块集成电路，使用时按照一定要求向显示模块发命令和写数据。

1．HD44780 液晶显示控制驱动器

HD44780 液晶显示控制驱动器是日立公司的产品，内有显示 RAM、字符发生器、振荡电路等，可实现单行 5×8、5×10 或双行 5×8 显示，有 80 条引脚，其中与微机连接的引脚是 DB0～DB7（数据线）、RS（命令寄存器/数据寄存器选择）、R/$\overline{W}$（读/写控制）和 E（芯片使能），工作电源 2.7V～6.5V。

HD44780 内有一个命令寄存器和一个数据寄存器。待显示的信息写入数据寄存器，而命令寄存器用来存放微机发来的命令，以规定或改变 HD44780 的工作方式或状态。主要命令格式如表 6.2 所示。

表 6.2　HD44780 的主要命令格式

功　　能	RS	$\overline{R/W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
清显示	0	0	0	0	0	0	0	0	0	1
光标回原位	0	0	0	0	0	0	0	0	1	—
进入模式设置	0	0	0	0	0	0	0	1	I/D	S
显示开关控制	0	0	0	0	0	0	1	D	C	B
光标或显示移动控制	0	0	0	0	0	1	S/C	R/L	—	—
功能设置	0	0	0	0	1	DL	N	F	—	—
读忙标志和地址	0	1	BF	AC	AC	AC	AC	AC	AC	AC

下面对表中的符号做简单介绍：

进入模式设置命令。I/D=1 表示增址，I/D=0 表示减址；S=1 表示伴随显示移动。

显示开关控制命令。D=0 表示关显示；C=0 表示关光标；D=0 表示不闪烁。

光标或显示移动控制命令。S/C=1 表示显示移动，S/C=0 表示光标移动；R/L=1 表示向右移动，R/L=0 表示向左移动。

功能设置命令。DL=1 选择数据总线为 8 位，与 8 位微机相连，DL=0 选择数据总线为 4 位；N=1 选择 2 行显示，N=0 选择 1 行显示；F=1 选择 5×10 模式，F=0 选择 5×8 模式。

读忙标志和地址。BF 为忙标志，BF=1 表示 HD44780 正在进行内部操作，不能接受命令，因此在向 HD44780 写入命令前，必须确保 BF=0；AC 为存储单元的地址。

2．HD44780 与MCS-51 单片机接口

如图 6.14 所示，用 74LS373 的 Q0 作为 RS 控制信号，芯片使能信号由单片机的读/写信号经过与非门产生，即不管是读操作还是写操作都选中 HD44780，单片机的 $\overline{RD}$ 经过反相后作为 HD44780 的 R/$\overline{W}$ 信号。因此 HD44780 的命令寄存器口地址为 FEH，数据寄存器的口地址为 FFH。

假设要求液晶显示器工作在 1 行 5×8 显示，则 HD44780 的初始化程序如下：

```
MOV    R0，#0FEH
MOV    A，#38H          ；8 位数据线，1 行，5×8 模式
MOVX   @R0，A
MOV    R2，#01H         ；清屏幕命令
ACALL  WRCMD            ；调写命令子程序
MOV    R2，#06H         ；进入模式为增量
```

```
        ACALL WRCMD
        MOV   R2，#0EH          ；开显示，显示光标，不闪烁
        ACALL WRCMD
WRCMD：MOV   R0，#0FEH
        MOVX  A，@R0
        JB    ACC.7，WRCMD      ；BF=1 则等待
        MOV   A，R2
        MOVX  @R0，A
        RET
```

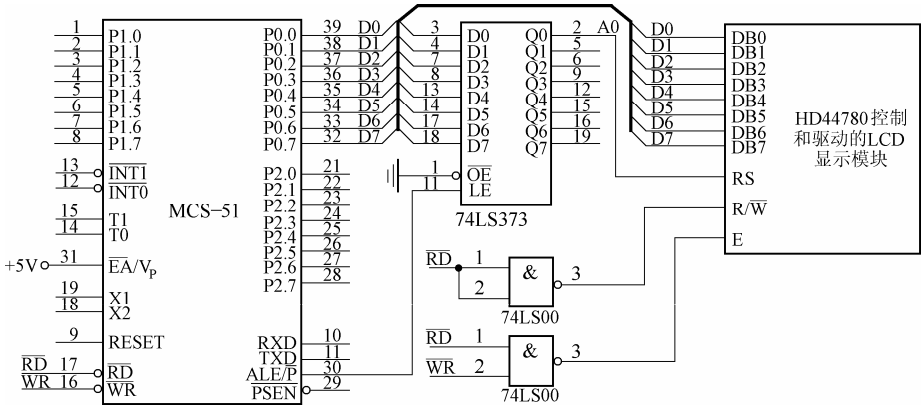


图 6.14　LCD 显示模块与单片机接口

初始化程序一般放在程序的开头，初始化完成以后就可以把要显示的数据写入 HD44780 的显示单元中。假设数据在 R2 中，写数据的程序段如下：

```
WRDAT：MOV   R0，#0FEH
        MOVX  A，@R0
        JB    ACC.7，WRDAT；BF=1 则等待
        MOV   R0，#0FFH
        MOV   A，R2
        MOVX  @R0，A
        RET
```

6.4　A/D转换器与单片机的接口技术

6.4.1　A/D转换器的性能参数与选型

1．A/D转换器的主要参数和意义

（1）量化误差与分辨率。A/D 转换器的分辨率习惯上以输出二进制数的位数或 BCD 码的位数表示。分辨率通常用百分数表示。例如一个 8 位二进制 A/D 转换器的分辨率为：

$$1/2^8\times100\%=1/256\times100\%=0.39\%$$

一个 4 位半 BCD 码 A/D 转换器的分辨率为：

$$1/19999\times100\%=0.005\%$$

量化误差和分辨率是统一的。量化误差是由于有限数字对模拟量进行离散取值而引起的误差。因此，量化误差理论上为一个单位分辨率，即±1/2LSB。提高分辨率可减少量化误差。

（2）转换精度。A/D 转换精度反映了一个实际 A/D 转换器在量化值上与理想 A/D 转换器进行模/数转换的差值，可以表示成绝对误差和相对误差。

不同厂家给出的精度参数可能不完全相同，有的给出综合误差，有的给出分项误差。分项误差包括：非线性误差、失调误差或零点误差、增益误差或标度误差、微分非线性误差等。

（3）转换时间与转换速率。转换时间被定义为 A/D 转换器完成一次转换所需要的时间，即从输入端加入信号到输出端出现相应数码的时间。转换速率是转换时间的倒数。

目前转换时间最短的 A/D 转换器为全并行 A/D 转换器，转换时间在 5～50ns，其次是逐次比较式 A/D 转换器，其转换时间也可达 0.4μs，再次是逐次逼近式，最后是双积分式。按照 A/D 转换器的转换时间分类：转换时间在 20μs 以下的为高速型；转换时间在 20～300μs 之间的为中速型；转换时间在 300μs 以上的为低速型。

（4）电源抑制比。电源抑制比（PSRR）反映 A/D 转换器对电源电压变化的抑制能力，用改变电源电压使数据发生±1LSB 变化时所对应的电源电压变化范围表示。

2．A/D转换器的选取原则

对于 A/D 转换器的选择来说，转换时间和分辨率是两个重要参数。特别是转换时间，应该考虑到高速 A/D 转换需要采样/保持电路，有时还需要缓冲放大器，这时若采用转换时间为 1μs 的转换器，当把系统中其他部件考虑进去时，转换时间已不再是 1μs，很可能是 10μs 或更长。因此，实际选择 A/D 转换器时通常要考虑以下问题：

（1）A/D 转换器用于什么系统？输出数据的位数是多少？系统应该达到多高的精度和线性度？

（2）提供给 A/D 转换器的输入信号范围多大？是单极性的还是双极性的？信号的驱动能力怎样？是否要经过缓冲滤波和采样/保持？

（3）对 A/D 转换器输出的数字代码及逻辑电平的要求如何？是二进制码还是 BCD 码，是串行还是并行？

（4）系统是在静态下工作还是在动态下工作？带宽为多少？采样速率为多少？

（5）参考电压是内部的还是外部的？是固定的还是变化的？

（6）A/D 转换器的工作环境如何？噪声、温度、振动等条件如何？

（7）电源电压、功耗、几何尺寸等其他因素。

6.4.2 ADC0809 与单片机接口

1．ADC0809 的结构及引脚特性

（1）ADC0809 的结构。ADC0809 是美国国家半导体公司生产的 8 位 A/D 转换器，它是采用逐次逼近的方法完成 A/D 转换。ADC0809 的内部结构如图 6.15 所示。ADC0809 由单一+5V 电源供电，片内带有锁存功能的 8 路模拟多路转换开关，可对 8 路模拟电压信号分时进行转换，完成一次转换约需 100μs。片内具有多路开关的地址译码器和锁存电路、高阻抗斩波器、稳定的比较器、256R 电阻 T 型网络和树状电子开关以及逐次逼近锁存器。输出具有 TTL 三态锁存缓冲器，可直接接到单片机数据总线上。

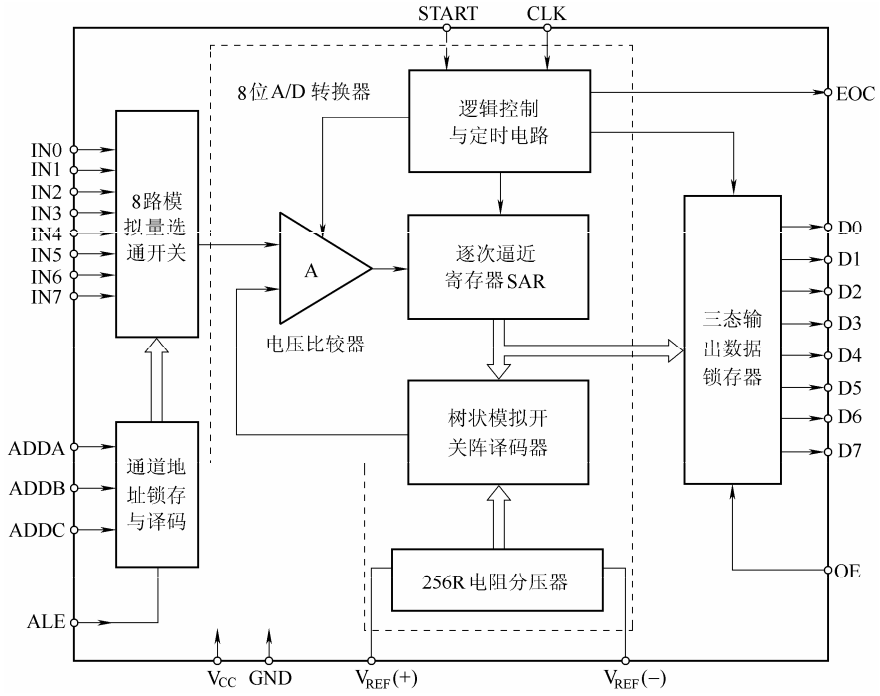


图 6.15　ADC0809 的内部结构

（2）ADC0809 的引脚特性。ADC0809 采用 28 脚双列直插式封装，引脚图如图 6.16 所示。各引脚的功能说明如下：

D0～D7：8 位数字量输出引脚。

IN0～IN7：8 路模拟量输入引脚。

V_{CC}：+5V 工作电源。

GND：地。

$V_{REF}(+)$：参考电压正端。

$V_{REF}(-)$：参考电压负端。

START：A/D 转换启动信号输入端。

ALE：地址锁存允许信号输入端。

EOC：转换结束输出引脚。开始转换时为低电平，转换结束为高电平。

OE：输出允许控制端。用以打开三态数据输出锁存器。

CLK：转换时钟信号。500kHz 左右。

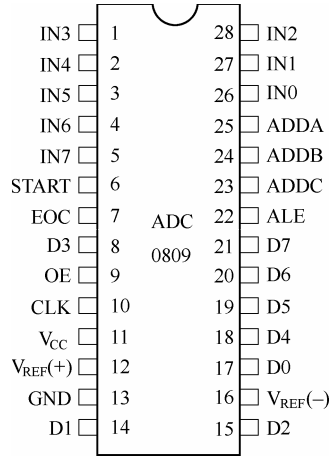


图 6.16　ADC0809 的引脚图

ADDA、ADDB、ADDC：地址输入线。经过译码后可选通 IN0～IN7 八个通道中的一个通道进行转换。A、B、C 的输入与被选通的通道的关系如表 6.3 所示。

表 6.3　ADDA、ADDB、ADDC 输入与被选通的通道的关系

被选通的通道	ADDC	ADDB	ADDA	被选通的通道	ADDC	ADDB	ADDA
IN0	0	0	0	IN4	1	0	0
IN1	0	0	1	IN5	1	0	1
IN2	0	1	0	IN6	1	1	0
IN3	0	1	1	IN7	1	1	1

（3）ADC0809 的工作时序。ADC0809 的时序图如图 6.17 所示。从时序图可以看出，通道地址由 ADDC、ADDB、ADDA 送入，在 ALE 升沿经锁存和译码选通一路模拟量。在 START 信号的下降沿，A/D 转换器开始转换，但经约 10μs 后，EOC 信号才变为低电平。当转换结束时，EOC 信号再变为高电平。当 ADC0809 接到 OE 信号变成高电平时，才将输出缓冲器的数字量有 D7～D0 输出。

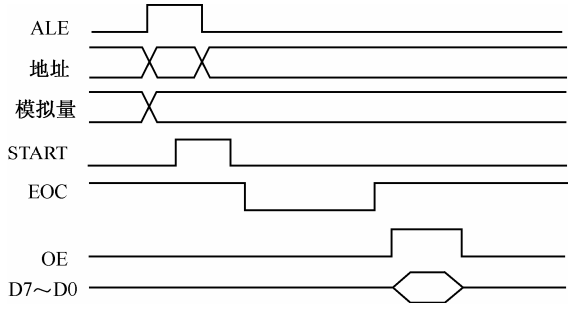


图 6.17　ADC0809 工作时序

（4）ADC0809 的主要性能指标。

分辨率 8 位。

不可调误差≤±1LSB。

转换时间 100μs。

温度范围−40℃～+85℃。

功耗 15mW。

单电源+5V 供电。

转换的模拟电压范围 0～5V。

2．ADC0809 与MCS-51 系列单片机接口

由于 ADC0809 的内部有一个三态数据输出锁存器和一个通道地址锁存及译码器，因此它与单片机的接口非常简单。如图 6.18 所示。

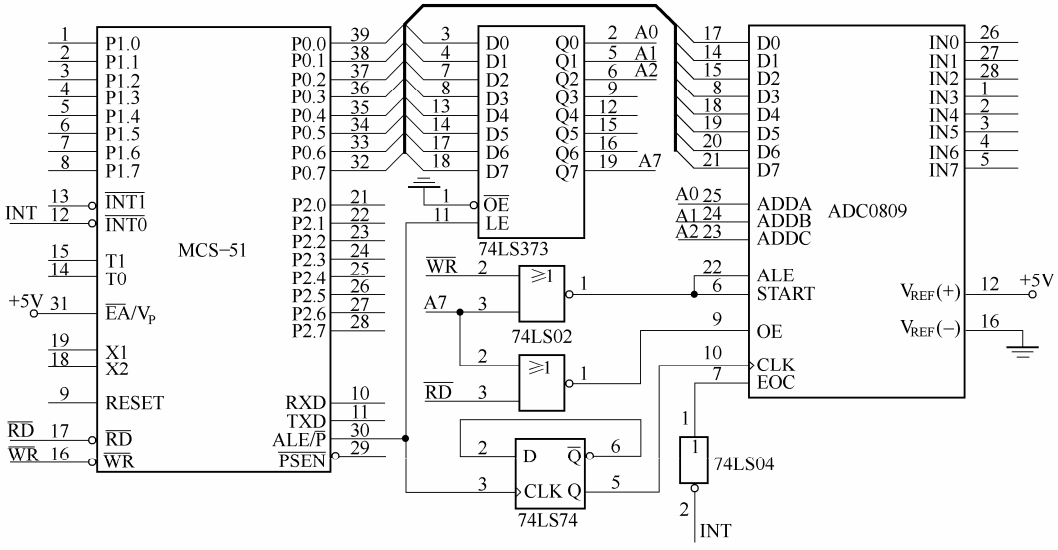


图 6.18　ADC0809 与单片机接口

为了尽量少占用单片机的资源，仅使用 P0 口，在编程时仅使用 MOVX　A，@Ri 指令或 MOVX　@Ri，A 指令，这样 P2 口就可以用作另外的用途，如前面讲的数码显示。P0 口经地址锁存器 74LS373 锁存以后得到 8 根地址信号，用 A0～A2 作为模拟量通道选择信号，A7 作为 ADC0809 的选择信号。由于系统采用 6MHz 的主频，所以将 ALE 信号进行二分频后就可以作为 ADC0809 的转换时钟信号。A/D 转换结束后，EOC 信号有效，通过反相后接至单片机的外部中断 0，这样既可以用查询方式进行数据采集，又可以用中断方式进行。

由图可知，8 路模拟量通道的通道地址为 78H～7FH，依次对应 IN0～IN7。

以图 6.18 所示的电路为例，将 IN0～IN7 通道的模拟量各采样一次，结果放入 40H～47H 单元中。下面分别用查询和中断两种方式实现。

（1）查询方式程序如下：

```
        ORG   0030H
START:  MOV   R0，#40H      ；采样数据存放首址
        MOV   R1，#78H      ；IN0 通道地址
        MOV   R2，#08H      ；模拟量通道数
        CLR   EX0
LOOP:   MOVX  @R1 ，A       ；启动 A/D 转换
        MOV   R3，#20H
DELY:   DJNZ  R3，DELY      ；等待 EOC 信号变低
        SETB  P3.2
POLL:   JB    P3.2，POLL    ；查询转换是否结束
        MOVX  A，@R1        ；读取转换结果
        MOV   @R0，A        ；存放结果
        INC   R0
        INC   R1
        DJNZ  R2，LOOP      ；8 通道未完，则采集下一通道
HERE:   SJMP  HERE
        END
```

在以上程序中需注意的是：在 A/D 转换被启动后，需 10μs 后 EOC 信号才变低，因此延时了一定时间才开始查询。

（2）中断方式程序如下：

```
        ORG   0000H
START:  AJMP  MAIN
        ORG   0003H
        AJMP  EXINT0
MAIN:   MOV   R0，#40H      ；采样数据存放首址
        MOV   R1，#78H      ；IN0 通道地址
        MOV   R2，#08H      ；模拟量通道数
        MOVX  @R1 ，A       ；启动 A/D 转换
        SETB  IT0           ；外部中断 0 为边沿触发方式
        SETB  EX0           ；允许外部中断 0 中断
        SETB  EA            ；开放 CPU 中断
```

```
HERE:   SJMP   HERE
EXINT0: PUSH   PSW          ；保护现场
        CLR    RS0
        CLR    RS1
        MOVX   A，@R1       ；读取转换结果
        MOV    @R0，A       ；存放结果
        INC    R0
        INC    R1
        DJNZ   R2，NEXT     ；8 通道未完，则采集下一通道
        CLR    EX0          ；采集完毕，则停止中断
        SJMP   DONE
NEXT:   MOVX   @R1 ，A      ；启动下一通道 A/D 转换
DONE:   POP    PSW
        RETI
        END
```

在以上的中断服务程序中，保护了 PSW 的内容，把寄存器的工作区域设置为 0 区，这是为了使程序在有复杂的主程序的情况下仍能使用。

6.4.3 AD574A与单片机接口

在某些单片机应用系统中，8 位 A/D 转换器不能满足精度的要求，需要选用分辨率更高的 A/D 转换器，主要有 10 位、12 位、16 位等 A/D 转换器。工程中用得最为广泛的是 12 位 A/D 转换器。MCS-51 单片机虽是 8 为单片机，但它可以同 12 位等高精度的 A/D 转换器连接。下面以 AD574A 为例介绍 12 位 A/D 转换器与 MCS-51 单片机的接口及程序设计方法。

1．AD574A的结构及引脚特性

（1）AD574A 的结构。AD574A 是一种逐次逼近式 12 位高速 A/D 转换器，它的内部结构如图 6.19 所示。从图中可以看出，AD574A 由模拟电路和数字电路两部分组成。其中模拟电路是由高性能的 AD565D/A 转换器和参考电压组成。数字电路由高性能的比较器、逐次逼近寄存器、三态输出数据锁存缓冲器和控制逻辑电路组成。

（2）AD574 的引脚特性。AD574 的引脚如图 6.20 所示。

D0～D11：12 位数字量输出。

CE：片选信号，高电平有效。

$\overline{CS}$：片选信号，低电平有效。

R/$\overline{C}$：数据读/启动信号。R/$\overline{C}$=1 时，读取转换结果；R/$\overline{C}$=0 时，启动 A/D 转换。

12/$\overline{8}$：输出数据长度选择信号。12/$\overline{8}$=1 时，D0～D11 上同时出现 12 位数字量；当 12 位数字量要分两次输出时，有12/$\overline{8}$和 A0 配合控制，12/$\overline{8}$=0 且 A0=0 时，D4～D11 输出；12/$\overline{8}$=0 且 A0=1 时，D0～D3 输出，此时 D4～D7 为 0，D8～D11 为高阻态。因此 AD574A 既可以同 8 位机相连，也可以同 8 位以上的计算机相连。

A0：字节选择信号。在R/$\overline{C}$=0 状态下，A0=0 启动 12 位 A/D 转换，A0=1 启动 8 位 A/D 转换。在R/$\overline{C}$=1 且12/$\overline{8}$=0 状态下，A0=0 读高 8 位数据，A0=1 读低 4 位数据。

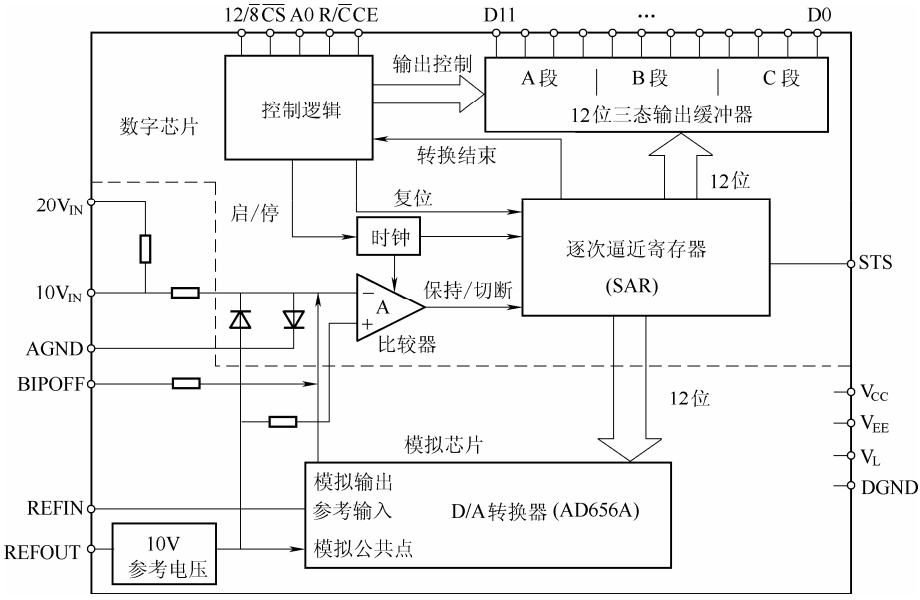


图 6.19　AD574A 的内部结构

为了方便理解，把以上逻辑控制信号归纳为表 6.4 所示。

表 6.4　AD574A 控制信号真值表

CE	$\overline{CS}$	R/$\overline{C}$	12/$\overline{8}$	A0	工作状态
0	×	×	×	×	禁止
×	1	×	×	×	禁止
1	0	0	×	0	启动 12 位转换
1	0	0	×	1	启动 8 位转换
1	0	1	1	×	12 位数据输出
1	0	1	0	0	高 8 位数据输出
1	0	1	0	1	低 4 位数据输出

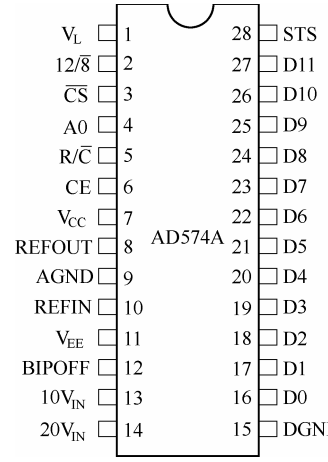


图 6.20　AD574A 的引脚图

STS：AD574A 的工作状态信号。STS=1 表示正处于转换状态；STS=0 表示转换完毕。

10VIN：10V 模拟电压输入。单极性时为 0～+10V，双极性时为−5V～+5V。

20VIN：20V 模拟电压输入。单极性时为 0～+20V，双极性时为−10V～+10V。

REFIN：参考输入，用于满量程调节。

REFOUT：内部 10V 参考电压输出。

BIPOFF：偏置输入，用于零点调节。

V_{CC}、VEE、VL：+15V、−15V、+5V 供电电源。

AGND：模拟地。

DGND：数字地。

（3）AD574A 的工作时序 AD574A 的工作时序如图 6.21 所示。

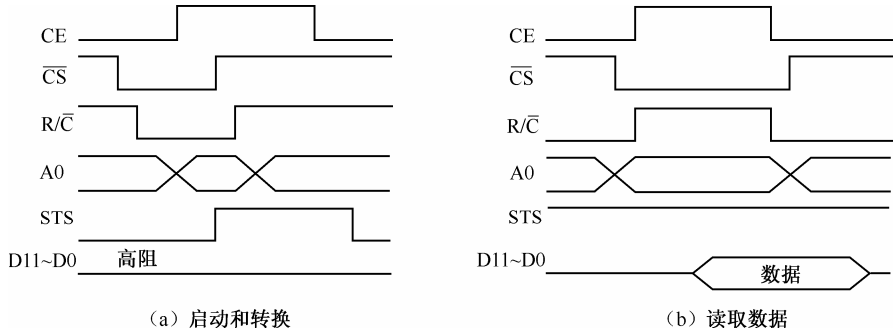


图 6.21　AD574A 的工作时序

（4）AD574A 的主要性能指标。

分辨率：12 位。

非线性误差：±1/2LSB。

模拟输入：单极性±5V；±10V。

供电电源：VL+4.5V～+6.5V；VCC+13.5V～+16.5V；VEE−13.5V～−16.5V。

功耗 450mW。

温度范围 0～+70℃。

转换时间 35μs。

（5）单极性和双极性电路。单极性电压和双极性电压均可直接与 AD574A 连接，但连接方式有所不同，如图 6.22 所示。图中电位器 W1 用于零点调整，W2 用于增益调整（即满量程调整）。

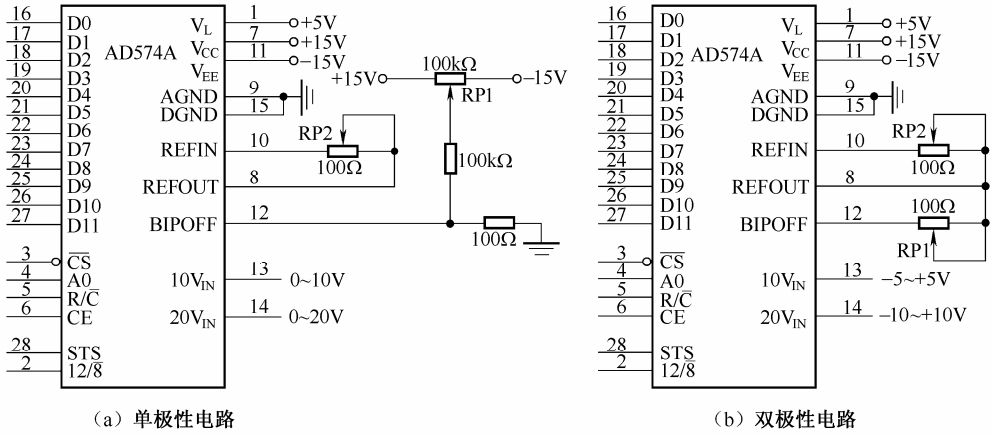


图 6.22　AD574A 的输入模拟电压连接

2．AD574A与MCS-51 系列单片机接口

如图 6.23 所示，由于 AD574A 的数字输出有三态锁存器，所以数据线直接与单片机的 P0 口连接。AD574A 的 12 位数据分两次输出，AD574A 的 D11～D4 与 P0.7～P0.0 相连，D3～D0 与 P0.7～P0.4 相连。$\overline{CS}$、A0 和 R/$\overline{C}$ 信号分别由地址信号 A7、A1 和 A0 相连。不管是读操作还是写操作都需要使能 AD574A，所以单片机的 $\overline{RD}$ 和 $\overline{WR}$ 信号经过与非门后与 CE 相连。同 8 位单片机相连，字长控制信号 12/$\overline{8}$ 固定接地；转换结束信号 STS 与 $\overline{INT0}$ 相连，用于状态查询。

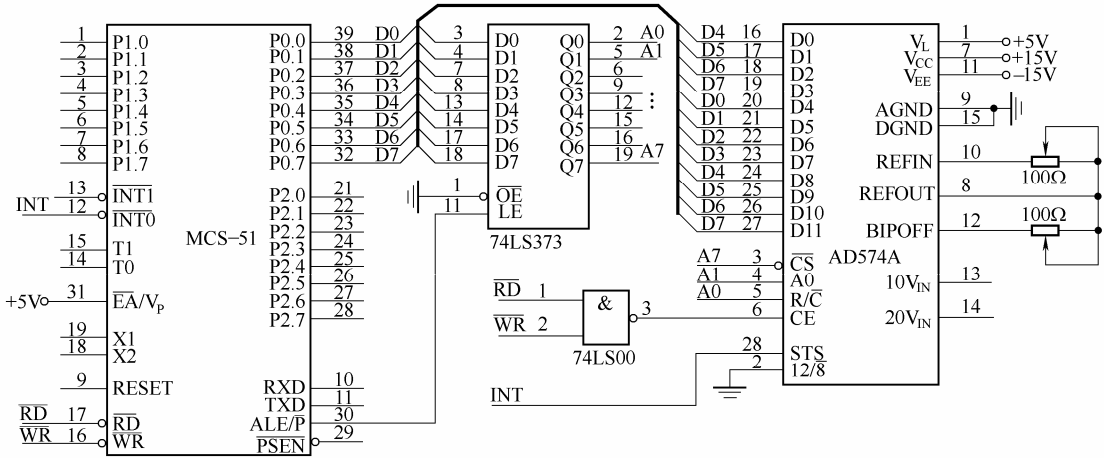


图 6.23　AD574A 与单片机接口

以图 6.23 为例，对模拟量电压采集一次，将结果放在单片机内部的 30H 和 31H 单元，要求 30H 单元放高 4 位，31H 单元放低 8 位。程序如下：

```
        ORG   0030H
        MOV   R0，#00H      ；A7、A1、A0 为低电平
        MOV   R1，#30H      ；结果单元地址
        MOVX  @R0，A        ；启动 A/D 转换
TEST：  JB    P3.2，TEST    ；查询转换是否完成
        MOV   R0，#01H      ；A7 和 A1=0、A0=1
        MOVX  A，@R0        ；读转换结果高 8 位
        MOV   @R1，A        ；存入 30H 单元
        MOV   R0，#03H      ；A7=0、A1 和 A0=1
        MOVX  A，@R0        ；读转换结果低 4 位
        XCHD  A，@R1        ；把结果的 D7～D4 位移至低 4 位
        SWAP  A             ；调整、拼装成低 8 位
        INC   R1
        MOV   @R1，A        ；存放低 8 位
        DEC   R1
        MOV   A，@R1
        SWAP  A             ；结果的 D11～D8 位移至字节低 4 位
        MOV   @R1，A        ；存放高 4 位
HERE：  SJMP  HERE
        END
```

以上是通过查询方式实现的数据采集，请读者思考，如何利用中断方式来实现，并写出相应的程序。

6.5 D/A转换器与单片机的接口技术

6.5.1 D/A转换器的性能参数与选型

1．D/A转换器的主要参数和意义

（1）分辨率：分辨率是指输入数字量发生变化时，所对应输出模拟量的变化量。在实际使用中，表示分辨率高低常用的方法是采用输入数字量的位数表示。

（2）转换精度。D/A 转换器的转换精度与 D/A 转换集成芯片的结构和接口配置的电路有关。在实际使用时，一般不考虑 D/A 转换误差，D/A 转换器的分辨率即为其转换精度。

（3）失调误差。当数字输入全为 0 时，模拟输出值与理论输出值的偏差即为失调误差。

（4）增益误差。实际转换的增益与理论增益之间的偏差值。

（5）温度系数。在规定的使用温度范围内，温度每变化 1℃，增益、零点、精度等参数的变化量。

（6）线性误差。模拟输出偏离理想输出的最大值。

（7）建立时间。输入数字量变化后，模拟输出量稳定到相应数值范围内所经历的时间。

2．D/A转换器的选取原则

（1）考虑性能指标的要求。根据给定工作条件，从静动态性能两方面考虑，在实际使用中主要考虑分辨率和建立时间。

（2）结构特性与应用特性的选择。数据格式是二进制码还是 2 的补码；是并行码还是串行码。数字电平是 TTL、CMOS 还是其他形式。

6.5.2 DAC0832 与单片机接口

1．DAC0832 介绍

（1）DAC0832 的内部结构。DAC0832 是美国 NS 公司的产品，它的内部结构如图 6.24 所示，它由 8 位输入寄存器、8 位 DAC 寄存器、8 位 D/A 转换器及控制电路组成。由于有两级锁存功能，因此它可实现双缓冲、单缓冲和直通工作方式。

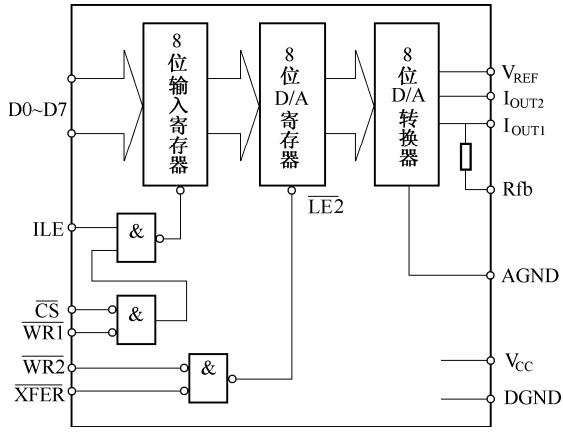


图 6.24　DAC0832 内部结构图

（2）DAC0832 的引脚图如图 6.25 所示，引脚功能如下：

D7～D0：8 位数据输入端。

ILE：输入寄存器锁存允许信号。

$\overline{CS}$：芯片选择信号。

$\overline{WR1}$：输入寄存器写信号。

$\overline{XFER}$：数据传送信号。

$\overline{WR2}$：DAC 寄存器写信号。

V_{REF}：基准电压，-10V～+10V。

Rfb：反馈信号输入端。

I_{OUT1}：电流输出 1 端。

I_{OUT2}：电流输出 2 端。

V_{CC}：电源。

AGND：模拟地。

DGND：数字地。

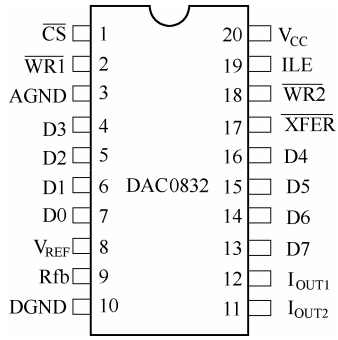


图 6.25　DAC0832 引脚图

2．DAC0832 与MCS-51 系列单片机接口

如图 6.26 所示，DAC0832 以单缓冲方式与单片机相连，用地址锁存器 74LS373 的 Q7 作为 DAC0832 的选择线，因此它的端口地址为 7FH，当 Q7=0，单片机的 $\overline{WR}$ 有效时，同时打开 DAC0832 的输入寄存器和 DAC 寄存器。以此为例，要求 DAC0832 输出锯齿波，周期自由的程序如下：

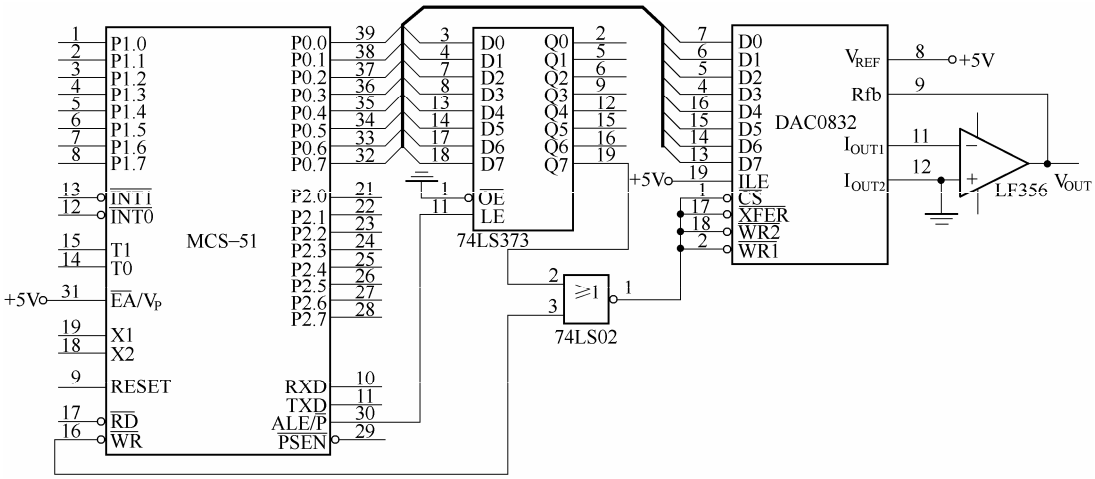


图 6.26　DAC0832 与单片机接口

```
        ORG    0030H
START:  MOV    R0，#7FH
        CLR    A
LOOP:   MOVX   @R0，A
        INC    A
        SJMP   LOOP
        END
```

请读者思考，要求以一定周期输出锯齿波，该怎样去实现它？又怎样去实现一定周期的正弦波信号？从而实现函数发生器功能。

6.5.3 DAC1210 与单片机接口

1. DAC1210 介绍

DA1210 是美国 NS 公司的产品，它的内部结构如图 6.27 所示，它与 DAC0832 相似。所不同的是输入寄存器由一个 8 位寄存器和一个 4 位寄存器组成，DAC 寄存器和 D/A 转换器都变成了 12 位。

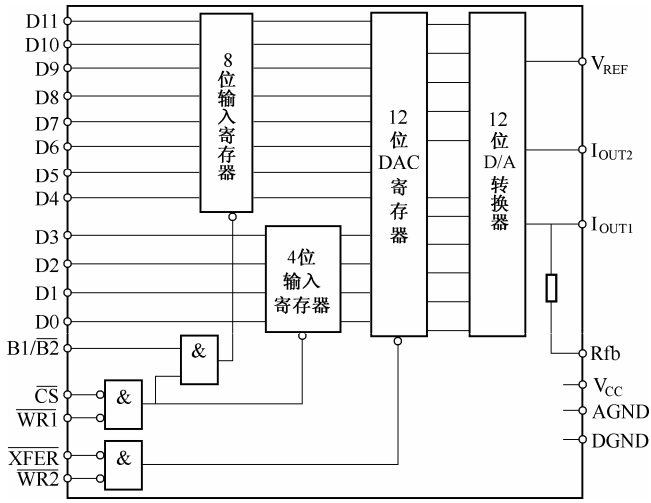


图 6.27　DAC1210 的内部结构图

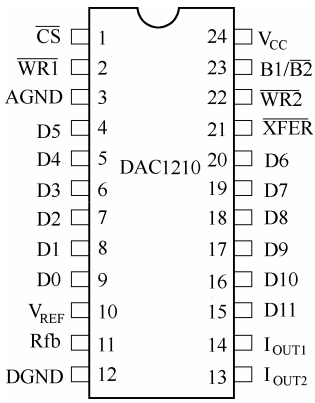


图 6.28　DAC1210 的引脚图

DAC1210 的引脚功能与 DAC0832 也非常相似，如图 6.28 所示，在此只把不同的引脚介绍一下。一是数据信号由 8 位增加为 12 位。二是 DAC0832 的 ILE 信号换成换成了 B1/$\overline{B2}$ 信号，当其为高电平时，同时可开通 8 位和 4 位输入寄存器，12 位数据均被写入；当其为低电平时，仅开通 4 位输入寄存器，写入 4 位数据。

2. DAC1210 与MCS-51 系列单片机接口

当DAC1210 与MCS-51 单片机连接时，数据需分两次写入，必须保证 12 位数据同时送入 D/A 转换器进行转换，这样才能使两次数据写入间隙输出不出现毛刺。

如图 6.29 所示，地址锁存器 74LS373 的 Q6 作为 DAC1210 的$\overline{CS}$控制信号，Q7 作为输入锁存器允许和传输控制信号。写入高 8 位时，Q6=0、Q7=1，此时高 8 位的低半字节也被 4 位输入寄存器锁存；Q6=0、Q7=0，写入低 4 位，同时也打通 12 位 DAC 寄存器，开始进行 D/A 转换。以此为例，要求 DAC1210 输出锯齿波，波形周期自由，程序如下：

```
        ORG     0030H
START:  MOV     R2，#0FFH       ；输出值高 8 位初值
```

```
        MOV   R3，#0F0H      ；输出值低 4 位初值
AGAIN： MOV   A，R2
        MOV   R0，#0BFH
        MOVX  @R0，A         ；输出高 8 位
        MOV   A，R3
        SWAP  A
        MOV   R0，#3FH
        MOVX  @R0，A         ；输出低 4 位
        CLR   C
        MOV   A，R3
        SUBB  A，#10H        ；输出值减一个单位
        MOV   R3，A
        MOV   A，R2
        SUBB  A，#00H
        MOV   R2，A
        ORL   A，R3
        JNZ   AGAIN          ；输出值不为 0，则继续
        SJMP  START          ；输出值为 0，重新开始
        END
```

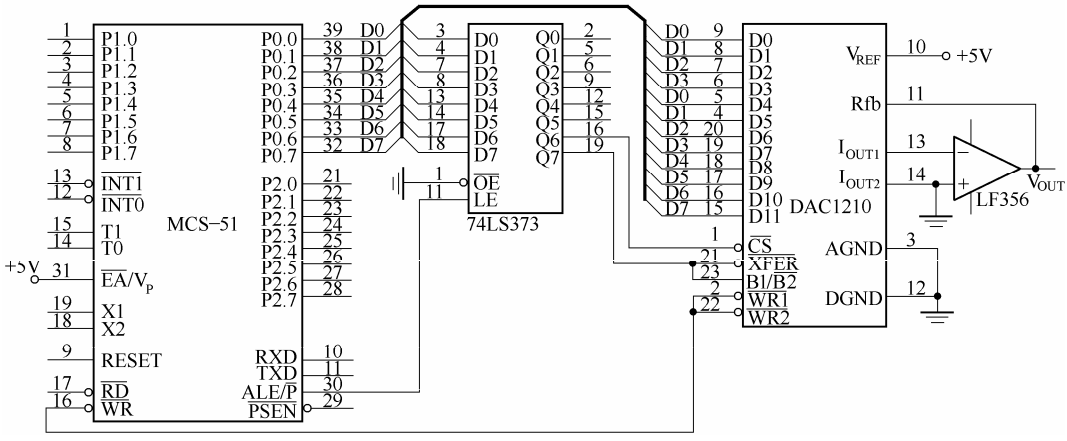


图 6.29　DAC1210 与单片机接口

6.5.4　V/I变换电路

在工程实际中，模拟量常常是以电流形式输出。电流输出有 0～10mA、0～20mA、4～20mA 等标准，由于采用恒流输出，因此信号不易受到干扰，传输的距离可以更远。

V/I 变换电路有多种多样，主要有集成运算放大器电路和专用集成电路两大类型。专用集成 V/I 变换电路使用简单，如 AD693 等。集成运算放大器 V/I 变换电路成本低廉，得到了广泛的应用，如图 6.30 所示。

图中 V_i 是待变换的模拟电压，一般是 0～5V 或 0～10V，I_o 为电流输出 4～20mA。当 $R_f << R_4+R_5$，$R_L << R_3+R_6$ 时，$V_1=V_2$，$V_o=I_oR_f$。

$$\frac{R_6}{R_3+R_6}V_i+\frac{R_3}{R_3+R_6}V_o=\frac{R_5}{R_4+R_5}V_b+\frac{R_3}{R_4+R_5}(V_f+V_o)$$

取 R_3=R_4=100kΩ、R_6=R_5=20kΩ，则

$$V_f=(V_i-V_b)R_5/R_3$$

$$I_o=V_f/R_f=(V_i-V_b)/5R_f$$

当 V_i 为 0～10V 时，取 V_b=−2.5V（由多圈电位器 RP_1 调整）。V_i=0V 时，调整 RP_1 使 I_o=4mA；V_i=10V 时，调整 RP_2 使 I_o=20mA。交替调整 RP_1 和 RP_2，直到满足精度要求为止。理论情况下 R_f=125Ω，实际应用中，取 R_7=100Ω，RP_2 为 100Ω的多圈电位器。

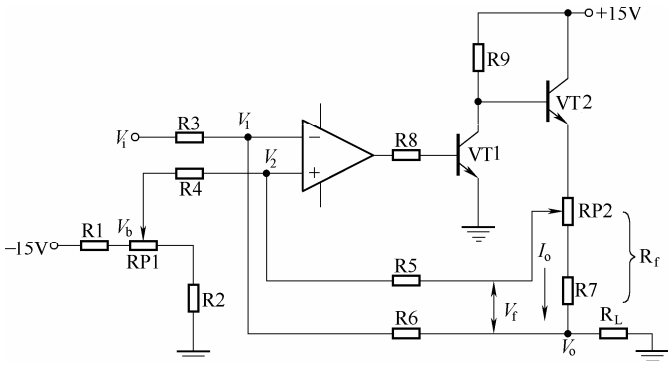


图 6.30　V/I 转换电路

6.6　串行接口技术

单片机与外部设备的连接有并行与串行接口两种方式。并行接口是指数据的各位同时进行传送，其优点是传输速度快，但随着传输距离的增加或速度的提高会导致干扰增加，产生数据传输错误。因 MCS-51 单片机的内部总线已引至引脚 P0～P3，因而它可以很方便地与并行接口芯片（如 ADC0809）进行并行数据传输。串行接口是指数据一位位地顺序传送，其特点是传输线路简单，只要少数几根传输线就可以实现双向通信，降低了成本，特别适用于远距离通信，但一般来说传送速度稍慢。常用的串行接口有 SPI、I^2C 和 CAN 等。

6.6.1　SPI串行总线

串行外围设备接口 SPI（Serial Peripheral Interface）是 Motorola 公司推出的一种具备高速（几 Mbps）、全双工三线式同步串行通信方式。由于硬件功能强，占用 CPU 时间少，适合传输较大量的数据。SPI 接口主要应用在 E^2ROM、FlashROM、实时时钟、A/D、D/A 转换器，还有数字信号处理器和数字信号解码器之间的数据传输等，设备连接如图 6.31 所示。

SPI 基本特性：

（1）采用主从工作方式（总线上只能有一个主机，可以允许多个从机），主机每次只能和一个从机进行通信。

（2）通信都由主机发起并负责提供 SCLK 同步时钟脉冲，每个时钟脉冲传递一位数据。

（3）两个数据信号，即负责数据发送的 SDO 和数据接收的 SDI。

（4）为保证通信正常，主机应采用和从机一致的通信协议（包括位发送顺序、数据长度、时钟极性和数据采样时刻等）。

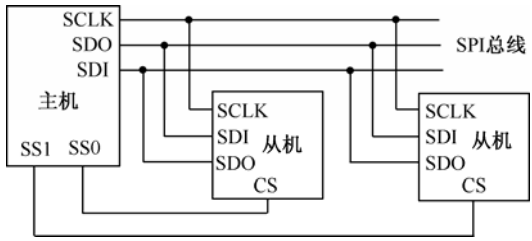


图 6.31　SPI 总线连接

6.6.2　SPI A/D、D/A转换器与MCS-51 单片机接口

串行接口器件以其封装小、低功耗、接口简单的特点，在现代电子产品设计中，得到日益广泛的应用。在此以 MAX1247 和 MAX525 为例，介绍 MCS-51 单片机下用软件虚拟 SPI 接口的方法。

1. MAX1247 简介

（1）MAX1247 结构。MAX1247 是美国 MAXIM 公司生产的具备 SPI 接口的高速、低功耗 12 位 A/D 转换芯片，其内部框图如图 6.32 所示。它是由一个 4 路模拟转换开关、采样保持器、A/D 转换器等构成，其输入的模拟信号类型(单/双极性，单端/差分输入)以及 A/D 转换时钟源（内部时钟/串行接口时钟）都可由进行软件设置，A/D 转换时间 6μs（2MHz 时钟），采用单一 2.7～5.25V 工作电压，A/D 转换参考电压需外接。

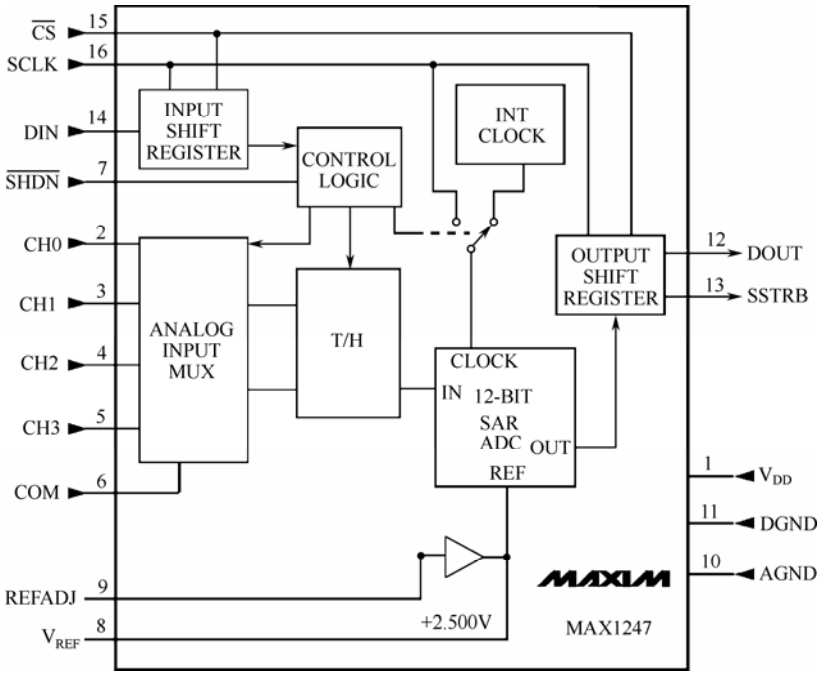


图 6.32　MAX1247 功能框图

（2）MAX1247 引脚特性。MAX1247 引脚图如图 6.33 所示。各引脚的功能说明如下：

CH0～CH3：4 路模拟信号输入端。

COM：模拟信号参考点。

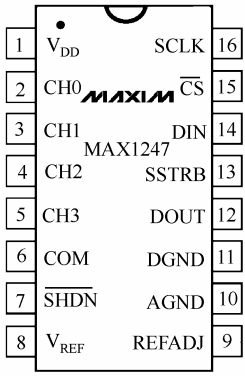


图 6.33　MAX1247 引脚图

$\overline{SHDN}$：三电平关断输入端。$\overline{SHDN}$ 接低电平，则使 MAX1247 进入关断模式，否则为正常工作模式；正常工作模式下，$\overline{SHDN}$ 接高电平或悬浮将影响内部参考电压放大器的补偿方式，接高电平为内补偿，浮地则为外补偿。

VREF：参考电压输出或 AD 参考电压输入端。

REFADJ：内部参考电压缓冲放大器输入端。

DOUT：串行数据输出端。在 SCLK 的下降沿时输出数据，$\overline{CS}$ 无效时输出为高阻。

SSTRB：串行 A/D 选通脉冲输出端。

DIN：串行数据输入端，在 SCLK 上升沿时锁存输入数据。

$\overline{CS}$：片选，低电平有效。

SCLK：串行时钟输入端。外部时钟模式下，SCLK 决定了 A/D 转换速度（占空比 40%～60%）。

VDD：电源。

AGND：模拟地。

DGND：数字地。

（3）MAX1247 工作时序。为启动 A/D 转换，必须按最高位（MSB）在前的方式传输 3 个字节。其中，第一个字节（控制字节）用来配置 MAX1247，其余字节用于接收转换结果。表 6.5 为控制字节格式，时序见图 6.34 和图 6.35 所示。

表 6.5　控制字节格式

BIT7 (MSB)	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0 (LSB)
START	SEL2	SEL1	SEL0	UNI/$\overline{BIP}$	SGL/$\overline{DIF}$	PD1	PD0

START：A/D 转换启动位，‘1’有效。

SEL2～SEL0：选择模拟输入通道。

UNI/$\overline{BIP}$：‘1’=单极性，‘0’=双极性。

SGL/$\overline{DIF}$：‘1’=单端输入，‘0’=差分输入。

PD1～PD0：“00”=全掉电模式，“01”=快速掉电模式，“10”=内部时钟，“11”=外部时钟。

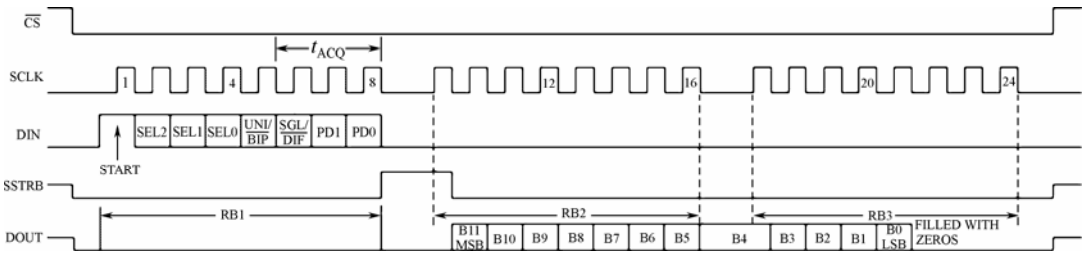


图 6.34　MAX1247 外部时钟模式下 SPI 接口时序

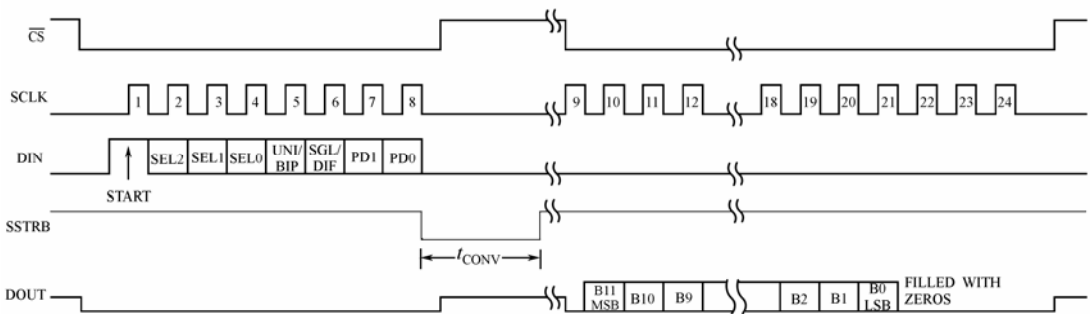


图 6.35　MAX1247 内部时钟模式下 SPI 接口时序

2. MAX525 介绍

（1）MAX525 基本描述。MAX525 是美国 MAXIM 公司生产的具备 SPI 接口的 12 位 D/A 转换芯片，内部框图见图 6.36 所示，由 4 个独立的采用 16 位双缓冲输入结构的低功耗、电压输出的 D/A 模块构成，4 路 D/A 可单独或同时更新。

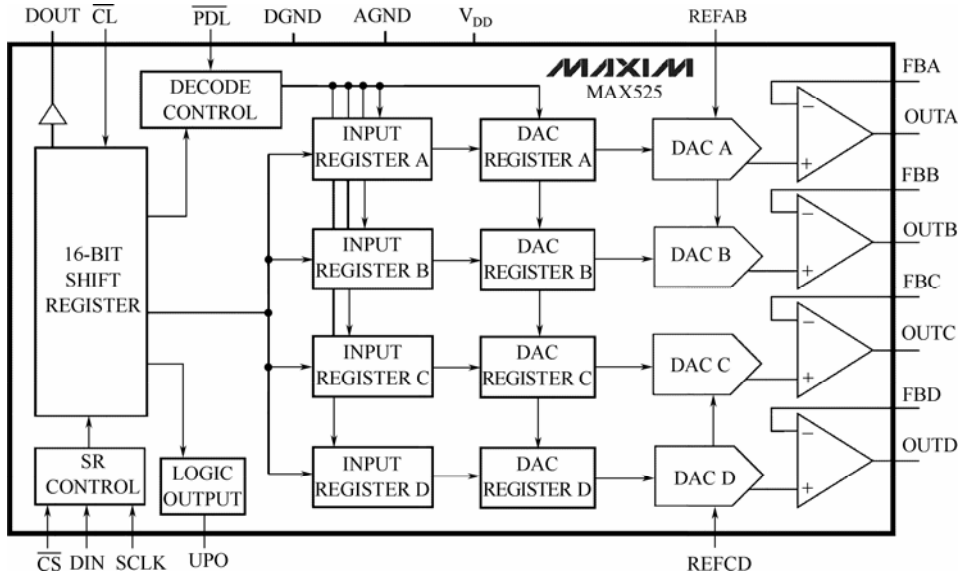


图 6.36　MAX525 功能框图

（2）MAX525 引脚特性。MAX525 引脚图如图 6.37 所示。各引脚的功能说明如下：

OUTD～OUTA：4 路 D/A 输出放大器输出端。

FBD～FBA：4 路 D/A 输出放大器反相输入端。

REFAB：D/A 转换器 A 和 B 的参考电压输入。

REFCD：D/A 转换器 C 和 D 的参考电压输入。

$\overline{CL}$：所有 D/A 转换器、寄存器以及输出信号清零,低电平有效。

UPO：用户可编程逻辑输出端。

$\overline{PDL}$：掉电保护，不使用芯片软件关闭功能。低电平有效。

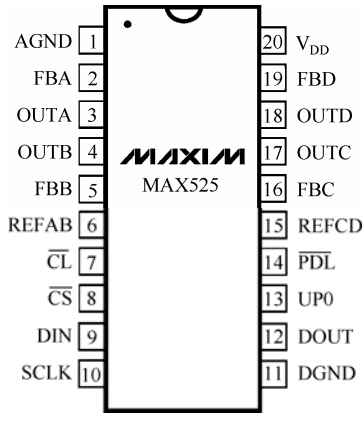


图 6.37　MAX525 引脚图

DOUT：串行数据输出端。

DIN：串行数据输入端，在 SCLK 上升沿时锁存输入数据。

$\overline{CS}$：片选，低电平有效。

SCLK：串行时钟输入端。

V_{DD}：电源。

AGND：模拟地。

DGND：数字地。

（3）MAX525 工作时序。MAX525 的控制命令为 16 位，它由 2 位 DAC 地址（A1，A0）和 2 位控制位（C1，C0）以及 12 位的数据位构成，按 MSB 在前的方式传输。命令字格式见表 6.6，时序见图 6.38 所示。

表 6.6　MAX525 命令格式

16 位串行控制字					功　能
MSB				LSB	
A1	A0	C1	C0	D11…D0	
0	0	0	0	12 位 DAC 数据	锁存进输入寄存器 A，不影响 DAC 寄存器值
0	1	0	1		锁存进输入寄存器 B，不影响 DAC 寄存器值
1	0	1	0		锁存进输入寄存器 C，不影响 DAC 寄存器值
1	1	1	1		锁存进输入寄存器 D，不影响 DAC 寄存器值
0	0	1	1	12 位 DAC 数据	锁存进输入寄存器 A，更新全部 DAC 寄存器
0	1	1	1		锁存进输入寄存器 B，更新全部 DAC 寄存器
1	0	1	1		锁存进输入寄存器 C，更新全部 DAC 寄存器
1	1	1	1		锁存进输入寄存器 D，更新全部 DAC 寄存器
0	1	0	0	任意值	用各自输入寄存器值更新相应 DAC 寄存器
1	0	0	0	12 位 DAC 数据	用输入移位寄存器值更新全部 DAC 寄存器
1	1	0	0	任意值	在 PDL 为“1”时，关闭芯片
0	0	1	0	任意值	UPO 输出“0”
0	1	1	0	任意值	UPO 输出“1”
0	0	0	0	任意值	空操作
1	1	1	0	任意值	模式 1， SCLKK 上升沿 DOUT 输出，全部 DAC 更新
1	0	1	0	任意值	模式 0， SCLKK 下降沿 DOUT 输出，全部 DAC 更新

3. MAX1247、MAX525 与MCS-51 接口

因 MCS-51 单片机硬件不支持 SPI 通信协议，在使用带 SPI 接口芯片时通常采用软件模拟 SPI 时序的方法。由 MCS-51、MAX1247 以及 MAX525 构成的模/数、数/模转换系统如图 6.39 所示。

图 6.39 中，P1.0 作 SPI 接口的 SCLK 输出，P1.1 和 P1.2 分别为 SDI 和 SDO，P1.3 和 P1.4 分别为 MAX1247 和 MAX525 片选，P1.5 为 A/D 转换选通脉冲 STRB。系统先进行 A/D 转换，完成后将 A/D 结果通过 D/A 转换器输出，以采集 0 通道模拟信号单极性、单端输入为例，将采集的数据放入单片机 40H 和 41H 单元，处理后存入 42H 和 43H 单元，然后将其从 D/A 转换器 A 通道输出。程序如下：

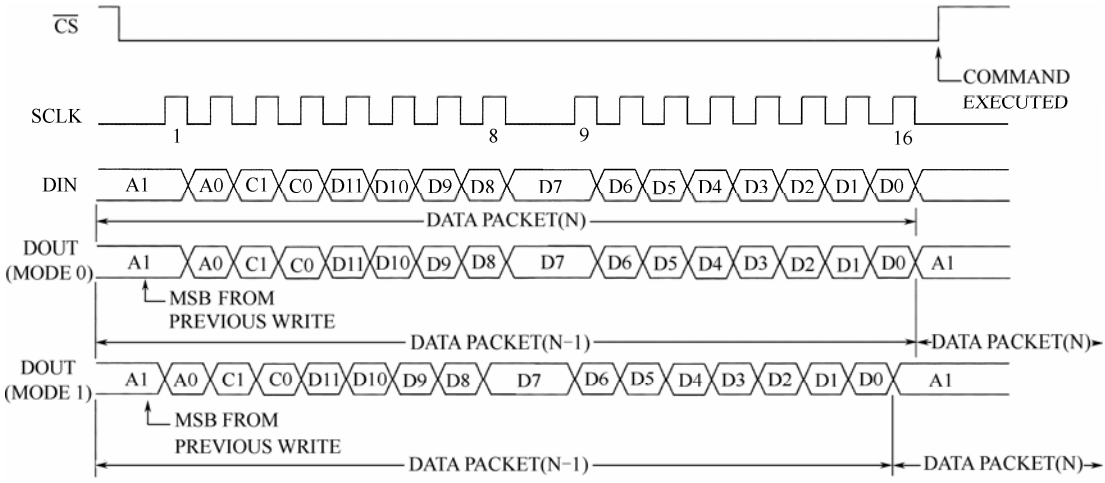


图 6.38　MAX525 SPI 接口时序

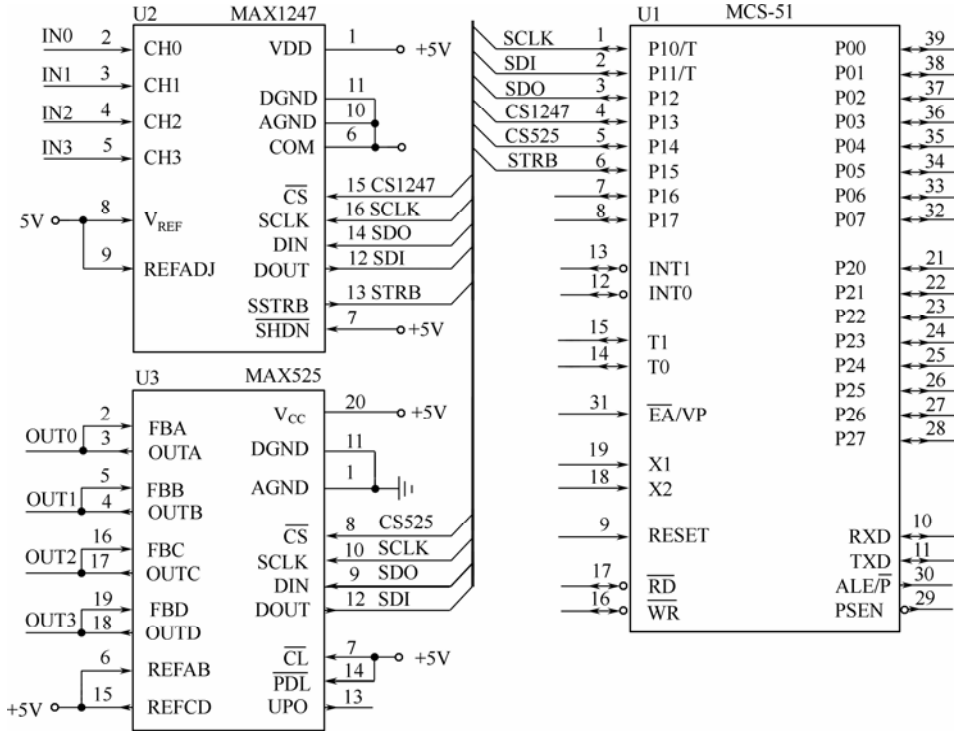


图 6.39　MAX1247、MAX525 与单片机接口

```
SCLK        BIT     P1.0
SDI         BIT     P1.1
SDO         BIT     P1.2
CS1247      BIT     P1.3
CS525       BIT     P1.4
STRB        BIT     P1.5
AD_H        EQU     40H
AD_L        EQU     41H
```

```
        DA_H        EQU   42H
        DA_L        EQU   43H
        AD_CTRL     EQU   9FH       ；CH0 单极性、单端输入，外部时钟模式
        DA_CHA      EQU   10H       ；DA 转换器 A
        DA_UPDATA   EQU   40H       ；更新全部 D/A 寄存器
        ORG         0030H
START： MOV         SP，#50H
        SETB        SDI
        SETB        CS1247          ；不使能 MAX1247
        SETB        CS525           ；不使能 MAX525
        CLR         SCLK            ；置 SCLK 为空闲状态
AD：    LCALL       STARTAD         ；启动 A/D 转换
        MOV         A，AD_L
        MOV         R0，#AD_H
        XCHD        A，@R0          ；交换 A/D 转换结果高低字节的低 4 位
        SWAP        A               ；交换高低 4 位
        MOV         AD_L，A         ；保存 A/D 转换结果低字节
        MOV         A，AD_H
        SWAP        A               ；交换高低 4 位
        MOV         AD_H，A         ；保存 A/D 转换结果高字节
        MOV         A，AD_H         ；将 A/D 转换值赋给 DA
        MOV         DA_H，A
        MOV         A，AD_L
        MOV         DA_L，A
        LCALL       WRDA            ；输出 D/A 值
        AJMP        AD
；SPI 接口字节读写子程序，A=SPI 接口输入/输出数据
SPI_RdWrBYTE：
        MOV         B，#08H
LOOP：  MOV         C，SDI          ；从 SDI 读入数据位
        RLC         A
        MOV         SDO，C          ；SDO 输出数据位
        NOP
        SETB        SCLK
        NOP
        CLR         SCLK
        DJNZ        B，LOOP
        RET
；MAX1247 控制子程序
STARTAD：
        CLR         CS1247          ；使能 MAX1247
```

```
        MOV     A，#AD_CTRL     ；设置 AD 转换器控制字
        LCALL   SPI_RdWrBYTE
        SETB    STRB
WAITAD：
        JNC     STRB，WAITAD    ；等待 A/D 转换结束
        SETB    SCLK            ；输出一个脉冲以便读取 A/D 值
        CLR     SCLK
；读入 AD 转换结果
        LCALL   SPI_RdWrBYTE
        MOV     AD_H，A         ；A/D 结果高字节地址 8 位
        LCALL   SPI_RdWrBYTE
        MOV     AD_L，A         ；A/D 结果低字节地址 4 位
        SETB    CS1247          ；不使能 MAX1247
        RET
；MAX525 控制子程序
WRDA：  CLR     CS525           ；使能 MAX525
        MOV     A，#DA_CHA      ；写 D/A 高 4 位到 MAX525 输入寄存器 A
        ORL     A，DA_H
        LCALL   SPI_RdWrBYTE
        MOV     A，DA_L         ；输出 D/A 低 8 位
        LCALL   SPI_RdWrBYTE
        MOV     A，#DA_UPDATA   ；更新 D/A 转换寄存器
        LCALL   SPI_RdWrBYTE
        CLR     A
        LCALL   SPI_RdWrBYTE
        SETB    CS525           ；不使能 MAX525
        RET
        END
```

6.6.3 I^2C串行总线

I^2C 总线（Inter Integrated Circuit Bus）产生于 20 世纪 80 年代，是一种由 PHILIPS 公司开发的两线式串行总线。I^2C 总线最初为音频和视频设备开发，通过对时钟和数据总线的双向控制，对电视机的各个功能模块（亮度、色度、对比度、音量等）进行有效的跟踪控制，提高电视的性能和稳定性，同时方便维修。现在，I^2C 的涵义已经拓展为各种嵌入式多媒体应用解决方案。

1. I^2C总线简介

I^2C 总线按照数据的传输速率分成三种：标准型（100Kbps）、快速型（400Kbps）和高速型（3.4Mbps），具有向前兼容能力。器件地址（7/10 位）由硬件设置，通过软件寻址来进行器件选择，系统扩展灵活。I^2C 总线传输速率有限，不能实现全双工通讯，因而不适合大量数据的传输。

2. I²C总线协议

I²C 总线协议有严格的时序要求。总线工作时，由时钟控制线 SCLK 传送时钟脉冲，由串行数据线 SDA 传送数据。总线信号分为地址和数据两部分，其中地址用来选择需要控制的电路。总线传送的每帧数据均为 1 个字节（8 位），但传送的字节个数没有限制，只要求每传送 1 个字节后，对方回应 1 个应答位（Acknowledge）。图 6.40 为 I²C 总线连接示意图，其完整的数据传送过程见图 6.41。在图 6.41 中：

（1）起始信号（START）。在时钟 SCLK 为高电平期间，数据线 SDA 出现由高电平向低电平的变化，启动 I²C 总线。

（2）应答信号（ACK）。I²C 总线的第 9 个脉冲对应应答位，对应 SDA 线上显示低电平时为“应答”信号，SDA 线上显示高电平时为“非应答”信号。

（3）数据位传送。I²C 总线开始信号或应答信号之后的第 1～8 个时钟脉冲对应一个字节的 8 位数据传送。在脉冲高电平期间，数据串行传送；在脉冲低电平期间，数据准备，允许总线上数据电平变化。

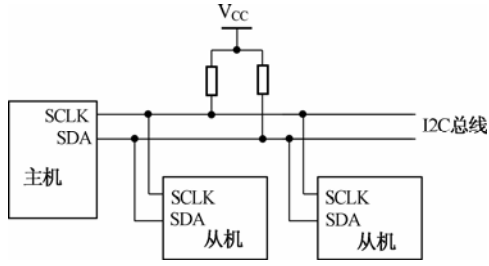


图 6.40　I²C 总线连接

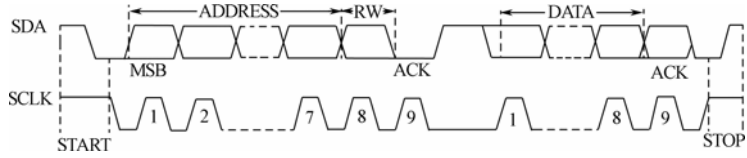


图 6.41　I²C 总线数据传送

3. I²C总线的接口技术

由于 I²C 总线支持多主系统，当系统中有多个 I²C 总线接口单片机时，会出现复杂的多主竞争情况，I²C 总线软硬件协议，以及 I²C 总线单片机中的 SFR 寄存器保证了多主竞争时的协调管理。因此，在多主的 I²C 总线系统中，为保证通信的可靠性必须使用带 I²C 总线接口的单片机；而在单主系统中，主器件永远占据总线，不会出现总线竞争，因而主器件可以选用没有 I²C 接口的器件，用两根 I/O 口线和相应软件来虚拟 I²C 总线接口，在时序满足要求后，就可以进行 I²C 总线操作，较常用的 I²C 总线虚拟软件有 VIIC1.0 等。

6.6.4　I²C器件与MCS-51 接口

1. 24C01 简介

24C01 是 Microchip 公司出产的带 I²C 接口的 1K 位电擦除 PROM，支持 100Kbps、400Kbps 两种通信速度，可编程次数达 100 万次，数据可保持 10 年时间，图 6.42 为器件内部框图。

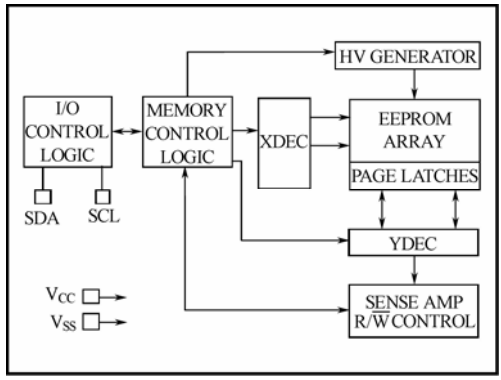


图 6.42　24C01 功能框图

2. 24C01 时序

主机对 24C01 的操作从主机发送包含 24C01 器件代码（1010）、读/写等信息的地址标识字节（或称控制字节）开始，而连接在 I^2C 总线上的 24C01 时刻监视着总线的变化情况，当器件地址匹配时处于非编程状态的 24C01 会发出应答位响应主机的操作，地址字节格式如下：

1	0	1	0	X	X	X	R/W

24C01 支持的写操作有单字节编程和页编程两种，其中字节编程步骤可简述为：主机在启动 I^2C 总线后随即发送控制字节，收到 24C01 应答信号后发送编程地址，在再次收到应答信号后发送 1 字节数据，在 24C01 应答后主机关闭 I^2C 总线，如图 6.43 所示。页编程的执行步骤和字节编程类似，只不过发送的是多个字节，但最多允许 8 字节。

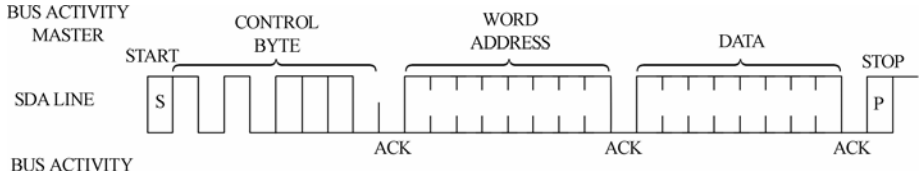


图 6.43　24C01 单字节写时序

24C01 的读操作有当前地址读、顺序读和随机读三种。以较复杂的随机读为例，主机操作步骤分为：主机为设置 24C01 的读地址首先向 24C01 发送执行写操作地址标识字节，然后是欲读取的单元地址，紧接着是读操作地址标识字节，最后接收 24C01 输出的数据，示意图见图 6.44。

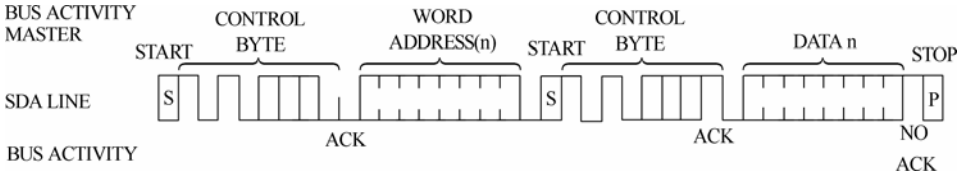


图 6.44　24C01 随机读操作时序

3. 24C01 与MCS-51 接口

MCS-51 单片机不具备 I^2C 接口，使用 I^2C 接口器件只能采用软件虚拟的方法，连接电路见图 6.45 所示。

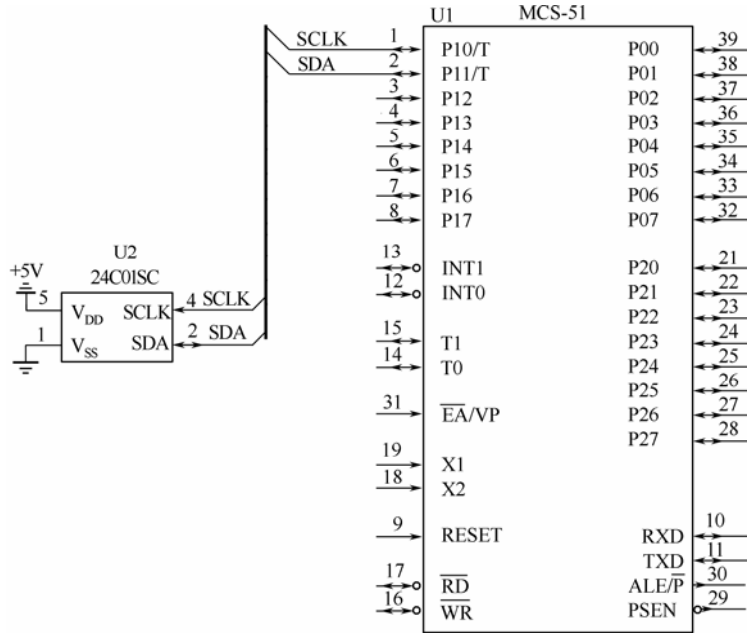


图 6.45　24C01 与 MCS-51 接口电路

在图 6.45 所示中，P1.0 作 I^2C 接口的 SCLK，P1.1 为 SDA。24C01 的单字节读写程序如下：

```
SCLK        BIT   P1.0
SDA         BIT   P1.1
I2CADDR     EQU   30H
I2Cdata     EQU   31H
            ORG   0030H
START:      MOV   SP，#50H
            SETB  SCLK              ；端口初始化
            SETB  SDA
            MOV   I2CADDR，#00H
            MOV   I2CData，#55H
            CALL  I2C_WRITE         ；写数据到 I²C
            MOV   I2CADDR，#00H
            CALL  I2C_READ          ；读数据
            MOV   I2CData，A
            SJMP  $
；写一个字节，I2CADDR 为 24C01 存储单元地址，I2Cdata 为待写入的数据
I2C_WRITE：LCALL  I2C_START         ；启动 I²C 总线
            MOV   A，#10100000B     ；发送写控制字 A0H
            LCALL I2C_WBYTE
            LCALL I2C_ACK
            JC    I2C_WRITE         ；=1，表示非应答，再次发送
            MOV   A，I2CADDR        ；设置数据地址
            LCALL I2C_WBYTE
```

```
                LCALL  I2C_ACK
                MOV    A，I2CData          ；发送数据到 I²C 总线
                LCALL  I2C_WBYTE
                LCALL  I2C_ACK
                LCALL  I2C_STOP            ；停止 I²C
                RET
；读一个字节，I2CADDR 为 24C01 存储单元地址，I²Cdata 读出的数据
I2C_READ：      LCALL  I2C_START           ；启动 I²C 总线
                MOV    A，#10100000B       ；发送写控制字 A0H
                LCALL  I2C_WBYTE
                LCALL  I2C_ACK
                JC     I2C_READ            ；=1，表示无确认，再次发送
                MOV    A，I2CADDR
                LCALL  I2C_WBYTE
                LCALL  I2C_ACK
I2C_READ1：     LCALL  I2C_START
                MOV    A，#10100001B       ；发送读控制字 A1H
                LCALL  I2C_WBYTE
                LCALL  I2C_ACK
                JC     I2C_READ1
                LCALL  I2C_RBYTE           ；接收数据
                MOV    I2CData，A
                LCALL  I2C_ACK
                LCALL  I2C_STOP            ；停止 I²C
                RET
；发送开始信号
I2C_START：     SETB   SCLK
                SETB   SDA
                NOP
                NOP
                CLR    SDA
                NOP
                NOP
                CLR    SCLK
                RET
；发送结束信号
I2C_STOP：      CLR    SDA
                NOP
                NOP
                SETB   SCLK
                NOP
```

```
                NOP
                SETB    SDA
                RET
；发送接收确认信号
I2C_ACk：       SETB    SDA
                SETB    SCLK
                NOP
                NOP
                JB      SDA，I2C_ACK0          ；读 ACK
                CLR     C
                SJMP    I2C_ACK1
I2C_ACK0：      SETB    C
I2C_ACK1：      CLR     SCLK
                RET
；送八位数据
I2C_WBYTE：MOV  B，#08H
I2C_WBYTE1：
                RLC     A
                MOV     SDA，C
                SETB    SCLK
                NOP
                NOP
                CLR     SCLK
                DJNZ    B，I2C_WBYTE1
                RET
；接收八位数据
I2C_RBYTE：MOV  B，#08H
                CLR     A
                SETB    SDA
I2C_RBYTE1：
SETB SCLK
                NOP
                NOP
                MOV     C，SDA
                RLC     A
                CLR     SCLK
                DJNZ    B，I2C_RBYTE1
                RET
                END
```

6.7 开关量输入/输出接口技术

前面讨论了模拟量接口技术，开关量接口也是 I/O 接口的重要组成部分。在计算机测控系统中，常常需要输入设备的工作状态和控制设备的启停，这些都需要通过开关量输入/输出接口来实现。

6.7.1 光电隔离技术和器件

在工业测控系统中，存在着强烈的干扰信号，必须将微机系统同现场设备隔离开来，实现强电与弱电的分离，切断干扰路径，以提高测控系统的抗干扰能力和工作可靠性。隔离措施主要有变压器隔离与光电隔离等，在开关信号或数字信号中，普遍采用光电隔离技术。

常用的光电隔离器是光电耦合器。光电耦合器的一次侧都是发光二极管，但二次侧有多种形式，如光敏二极管、光敏晶体管等，因而就有二极管-二极管光电耦合器、二极管-晶体管光电耦合器、二极管-达林顿管光电耦合器等，如图 6.46 所示。

选用光电耦合器主要考虑一次侧的承受电压、一次侧的承受电流、二次侧的承受电压、二次侧的承受电流、信号的建立时间等参数。常用的光电耦合器有 TLP 系列（如 TLP521-1/-2-/-3/-4 等）、4N 系列（如 4N25/29/30/35/45 等）、TIL 系列（如 TIL113 等），在高速场合下常用 6N137 等。

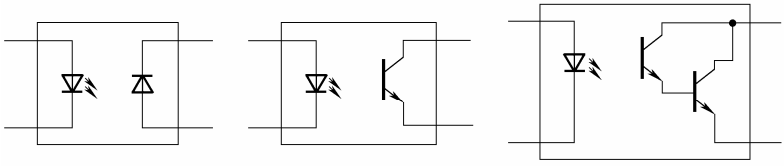


图 6.46　光电耦合器内部结构类型图

6.7.2 开关量输入接口

开关量输入信号在实际应用中主要有按钮、行程开关、接近开关等，在应用时有不带电（干接点）和带电（湿接点）两种情况。在带电时（如交流 220V），通常用继电器将其转换为干接点，然后由微机的供电电源对其供电，满足 TTL 电平标准。对于不带电情况，由于现场距离比较远，因此常用直流电源（24V、12V 等）对其供电，为了满足 TTL 电平要求，必须进行电平转换，常用的转换电路是用光电耦合器实现，如图 6.47 所示。

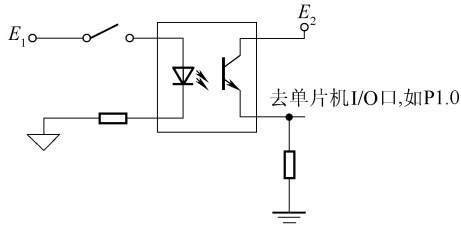


图 6.47　开关量输入电路

6.6.3 开关量输出接口

开关量输出常带动功率设备，控制设备的启停。输出形式有继电器、可控硅、晶体管等，继电器属于触点输出，而可控硅和晶体管则是无触点输出。由于是功率输出，因此应用时必须采取光电隔离措施以消除干扰。

继电器输出形式如图 6.48 所示。固态继电器（SSR）实为可控硅和晶体管输出形式，SSR 是无触点输出。固态继电器又分为直流固态继电器（DCSSR）和交流固态继电器（ACSSR）两种，DCSSR 的内部结构如图 6.49 所示，ACSSR 的内部结构如图 6.50 所示。

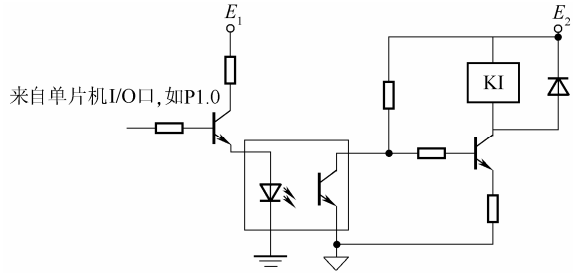


图 6.48　继电器输出电路

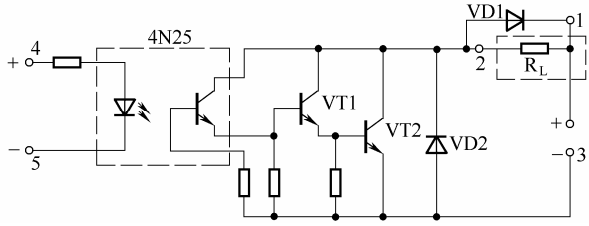


图 6.49　DCSSR 内部结构图

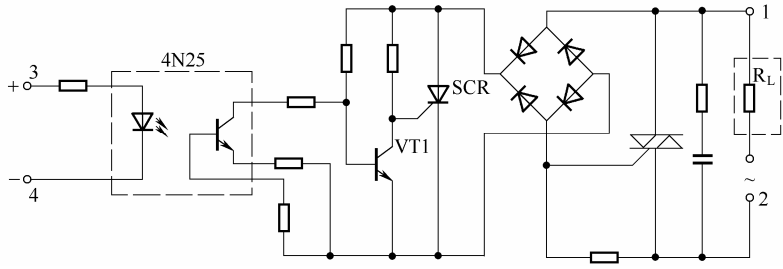


图 6.50　ACSSR 内部结构图

由图可知，SSR 内部具有光电耦合电路，单片机输出的 TTL 电平或 CMOS 电平可直接与 SSR 连接。对于 DCSSR，控制输入接至 4、5 端，直流负载接 1、2 端，负载的供电电源接 1、3 端。对于 ACSSR，控制输入接至 3、4 端，交流负载与交流供电电源一起接至 1、2 端。

常用的固态继电器如表 6.7 所示。

表 6.7　常用固态继电器特性参数

参数名称		单位	DCSSR		ACSSR	
			CCJ-1DD	C603 系列	CIJ-2.5AP	CG3C 系列
输入侧	控制电压	V	6～30	5～15	3～30	5～15
	控制电流	mA	3～30	3～32	＜30	3～32
	爆界接通电压	V		≤3		≤2.8
	爆界关断电压	V		≥1		≥1.8
	工作电压	V	24	30～180	30～220	140～400
	工作电流	A	1	1～10	2.5	1～20
	断态漏电流	mA	0.01	＜56	＜5	＜5
	通态压降	V	1.5	＜2	1.8	＜1.5
	过零电压	V			±15	
	导通时间	ms	200	50		10
	关断时间	ms	1	0.1		10
	单位功耗	W/A		1.5		1.25

续表

参 数 名 称	单位	DCSSR		ACSSR	
		CCJ-1DD	C603 系列	CIJ-2.5AP	CG3C 系列
输入绝缘电阻	Ω	10^9	10^9	10^9	10^9
输入绝缘电压	V		2500		2500
外形尺寸	mm^3	35×25×14	35×25×14	42×30×20	42×30×20

本 章 小 结

接口技术是本课程的核心内容之一。本章要求：掌握键的机械特性与抖动的软硬件消除方法；掌握独立式键盘与矩阵式键盘的结构特点，以及键的识别方法与过程；了解数码管的共阴、共阳结构，掌握静态显示与动态显示方式，特别是动态扫描显示原理及其实现程序；在键盘与显示接口电路中，推崇使用单片机的基本 I/O 口，在基本口不够时可通过扩展 I/O 来实现；了解液晶显示方式与液晶显示器与单片机的连接；了解 A/D 转换器与 D/A 转换器的性能指标与选取原则；掌握 ADC0809 的性能，与 MCS-51 单片机如何连接，通道地址如何确定，如何以无条件方式、查询方式和中断方式采集模拟信号；掌握 12 位 A/D 转换器 AD574A 的性能，以及它与 MCS-51 单片机如何进行连接与数据采集；掌握 DAC0832 和 DAC1210 性能，它们如何与 MCS-51 单片机的连接，单缓冲方式与双缓冲方式的性能特点等；掌握 SPI 和 I^2C 串行总线接口技术；掌握干湿接点的输入，继电器、固态继电器等开关接点的输出形式。

习 题 6

6.1 为什么要消除键盘的机械抖动？有哪些方法？

6.2 独立式键盘和矩阵式键盘各有什么特点？分别用在什么场合？

6.3 如图图 6.4 所示的独立式键盘，试写出定时中断的键盘扫描程序。

6.4 LED 静态显示和动态显示方式各有什么优缺点？

6.5 动态显示的原理是什么？

6.6 采用 8155 并行扩展接口设计一个 2×8 键盘和 8 位 LED 显示电路，并编写相应的键盘扫描程序和动态显示程序。

6.7 用单片机内部定时器来控制对模拟量信号的采集。如图 6.18 所示，设系统时钟为 6MHz，要求每分钟采集一次模拟信号，写出对 8 路模拟信号采集一遍的程序。

6.8 用 ADC0809 设计一个数据采集系统，要求 8 个通道的地址为 7FF8H～7FFFH，每 10ms 采样一路模拟量信号，每路信号采样 8 次，采的数据放在外部 RAM3000H 开始的存储单元中。设系统时钟为 6MHz，请编写出相应的程序。

6.9 用 DAC0832 设计一个模拟量输出接口，端口地址为 FEFFH，要求其产生周期为 5ms 的锯齿波。设系统时钟为 6MHz，请编写出相应的程序。

6.10 用 MAX525 设计一个模拟量输出接口，SPI 信号连接见图 6.39，要求其产生周期为 20ms，占空比为 10%的矩形波。设系统时钟为 6MHz，请编写出相应的程序。

6.11 用 24C01 实现 4 字节汽车行驶里程（km）备份，I^2C 信号连接见图 6.45，ROM 备份首地址为 10H，要求备份间隔时间为 1 分钟。设系统时钟为 6MHz，请编写出相应的程序。

6.12 设计一个用单片机实现的三相电动机正反转电路，设有正反转按钮，试画出单片机控制电路和电机主回路图，并编写相应的控制程序。

第 7 章　MCS-51 系列单片机串行通信

本章主要内容

串行通信的基本概念。MCS-51 系列单片机串行口的组成及其编程。串行通信接口标准。PC 机与单片机通信。

7.1　串行通信的基本概念

7.1.1　数据通信

在实际工作中，计算机的 CPU 与外部设备或者两台计算机之间常常要进行信息交换，所有这些信息交换均可称为通信。

通信方式有两种，即并行通信和串行通信。通信距离比较近时往往采取并行通信方式，传输效率高；通信距离比较远时往往采取串行通信方式，传输成本低。并行通信是指数据的各位同时进行传送的通信方式，数据有多少位，就需要多少根传送线。串行通信是指数据一位一位按顺序传送的通信方式，它只需要一对传输线，特别适合远距离通信。

7.1.2　串行通信的传输方式

串行通信的传输方式通常有三种：一种为单工方式，这种方式只允许数据从一个设备发送给另一个设备，即数据传送是单向的；第二种为半双工方式，这种方式既允许数据经同一信道从甲设备传送给乙设备，又允许数据从乙设备传送给甲设备，但不能同时进行数据的发送和接收；第三种为全双工方式，这种方式要求通信双方拥有两个独立信道，不仅数据传送是双向的，而且发送和接收可以同时进行。

7.1.3　异步通信和同步通信

1．异步通信

在异步通信中，数据是一帧一帧（包括一个字符代码或一字节数据）传送的，每一帧数据的格式如图 7.1 所示。

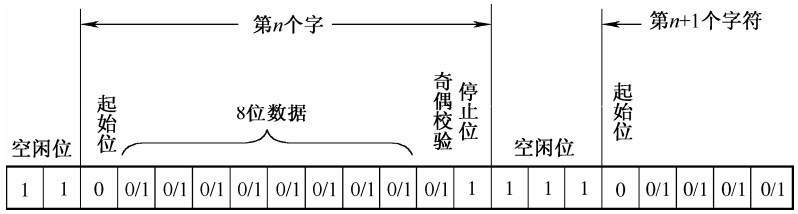


图 7.1　异步通信数据帧格式

在帧格式中，一个字符由四个部分组成：起始位、数据位、奇偶校验位和停止位。首先是一个起始位（一个周期低电平），然后是 5～8 位数据位（规定低位在前，高位在后），接下来是奇偶校验位（可省略），最后是停止位（一个周期以上高电平）。起始位（低电平）信号只占用 1 位，用来通知接收设备一个待接收的字符开始到达。线路上不传送字符时始终保持为 1，表示空闲。接收端不断检测线路的状态，若连续为 1 后又检测到一个 0，就知道来了一个新字符，应马上准备接收。帧信息的起始位还被用作同步接收端的时钟，以保证以后的接收能正确进行。

起始位后面紧接着是数据位，数据可以是 5 位（D0～D4）、6 位、7 位或 8 位（D0～D7）。

奇偶校验位（D8）只占 1 位，但在字符中也可以规定不用奇偶校验位，这一位就可以省去。也可用这一位来表示这一帧中的字符信息的性质（地址/数据等）。

停止位用来表示字符的结束，它一定是逻辑 1。停止电平可以是 1 位、1.5 位或 2 位。接收端收到停止位后，即表示上一字符传送完毕，同时，也为接收下一个字符做好准备。

异步通信的传输速率可达 20kb/s。

2．同步通信

同步通信中，在数据开始传送前用同步字符来指示，常用 1～2 个同步字符，并由时钟来实现发送端和接收端同步，即检测到规定的同步字符后，下面就连续按顺序传送数据，直到通信告一段落。同步传送时，字符之间没有间隙，也不用起始位和停止位，仅在数据块开始时用同步字符 SYNC 来指示，其数据格式如图 7.2 所示。

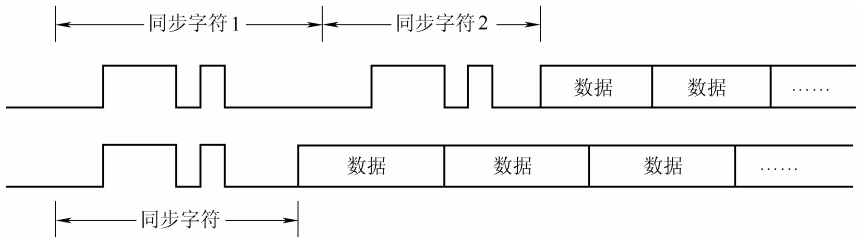


图 7.2　同步传送的数据格式

同步字符的插入可以是单同步字符或双同步字符，然后是连续的数据块。同步字符可以由用户约定，当然也可以采用 ASCII 码中规定的 SYNC 代码，即 16H。按同步通信时，先发送同步字符，接收方检测到同步字符后，即准备接收数据。

在同步传送时，要求用时钟来实现发送端与接收端之间同步。为了保证接收正确无误，发送方除了传送数据外，同时还要传送时钟信号。

同步通信的传输速率可达 56kb/s 或更高。

3．波特率

波特率，即数据传输速率，表示每秒钟传送二进制代码的位数，它的单位是 b/s。波特率对于 CPU 与外界的通信是很重要的。假设数据传送率是 120 字符/秒，而每个字符格式包含 10 个代码位（1 个起始位、1 个停止位、8 个数据位），这时，传送的波特率为：

10b/字符×120 字符/s=1200b/s

每一位二进制代码的转送时间即为波特率的倒数。异步通信的传输速率在 50b/s～19200b/s 之间。

7.2 MCS-51 系列单片机串行通信接口

MCS-51 系列单片机内部有一个功能很强的全双工串行异步通信接口，它既可以作为 UART 使用，能方便地构成双机或多机通信系统，也可以在外接移位寄存器后扩展并行 I/O 口。

7.2.1 串行口的结构与组成

图 7.3 所示为 MCS-51 系列单片机的串行口结构示意图。它主要由两个串行数据缓冲器（SBUF）、发送控制器、发送端口、接收控制器和接收端口等组成。串行口的工作方式和波特率由专用寄存器 SCON 和 PCON 控制。

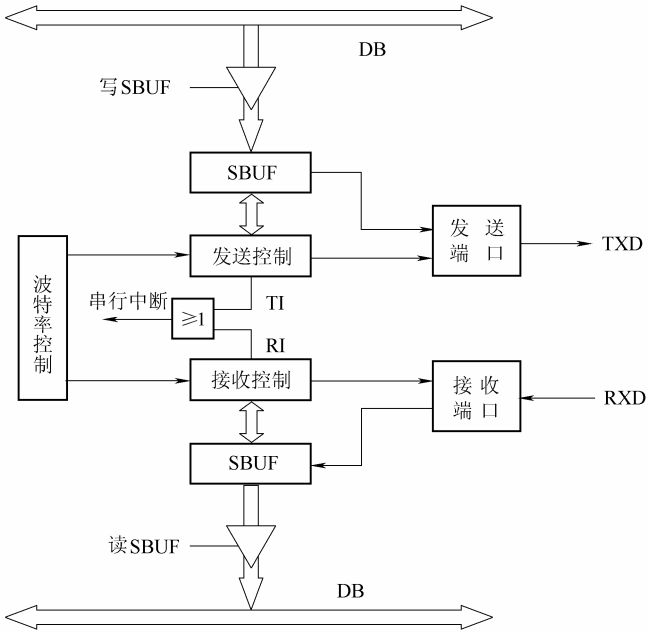


图 7.3 MCS-51 单片机串行口结构框图

1．串行数据缓冲器（SBUF）

MCS-51 系列单片机串行口有两个串行数据缓冲器，一个用于发送数据，另一个用于接收数据，可以同时用来发送和接收数据。发送缓冲器只能写入，不能读出。接收缓冲器只能读出，不能写入。两个缓冲器使用同一符号 SBUF，共用一个地址 99H，根据读、写指令来确定访问其中哪一个。

发送数据时，执行一条将数据写入 SBUF 的传送指令，即可将要发送的数据按事先设置的方式和波特率从 TXD 端串行输出。一个数据发送完毕后，串行口能向 CPU 提出中断请求，发送下一个数据。

接受数据时，当一帧数据从 RXD 端经过接收端口（移位寄存器）全部进入 SBUF 后，串行口发出中断请求，通知 CPU 接收这一数据。CPU 执行一条读 SBUF 的指令，就能将接收的数据送入某个寄存器或存储单元。与此同时，接收端口接收下一帧数据。为了避免前后两帧数据重叠，接收器是双缓冲的。

2．串行口控制寄存器（SCON）

SCON 用于控制串行口的工作方式，同时还包含要发送和接收到的第 9 位数据及串行口中断标志位。该寄存器的字节地址为 98H。各位的定义如下：

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0、SM1：串行口工作方式选择位，由软件设定。共有四种方式，见表 7.1。表中 f_{osc} 是 MCS-51 的振荡频率。

表 7.1　串行口工作方式

SM0	SM1	方式	功 能 说 明
0	0	0	移位寄存器输入/输出，波特绿为 $f_{osc}/12$
0	1	1	8 位 UART，波特率可变（T1 溢出率/n，n=32 或 16）
1	0	2	9 位 UART，波特率为 f_{osc}/n，n=64 或 32）
1	1	3	9 位 UART，波特率可变（T1 溢出率/n，n=32 或 16）

SM2：多机通信控制位，由软件设定。串行口的方式 2 和方式 3 适用于多机通信。在方式 2 或方式 3 中，当 SM2=1，若接收到的第 9 位数据（RB8）为 0，则不能置位 RI；只有收到 RB8=1，才置位 RI。SM2=1 用于多机通信中，只接收地址帧，不接收数据帧。而当 SM2=0 时，只要接收到一帧信息（无论是地址还是数据），RI 都被置位。双机通信时，通常使 SM2=0。在方式 0 中，SM2 必须为 0。

REN：允许接收控制位，由软件设定。REN=1 时允许接收，REN=0 时禁止接收。

TB8：方式 2 和方式 3 中要发送的第 9 位数据，由软件设定，用作奇偶校验位或地址/数据标志位，后者多用于多机通信。

RB8：方式 2 和方式 3 中接收到的第 9 位数据，在方式 1 中，如果 SM2=0，则 RB8 为收到的停止位。方式 0 不使用 RB8。

3．电源控制寄存器（PCON）

PCON 主要用于 CMOS 型单片机的低功耗电源控制，其字节映像地址为 87H，其格式如下：

SMOD	—	—	—	GF1	GF0	PD	IDL

低 4 位用于电源控制，与串行接口无关，在第 1 章已经学习过了。最高位 SMOD 用于串行口波特率控制，SMOD=1 时波特率增大一倍。

7.2.2　串行口的工作方式

如前所述，MCS-51 单片机的串行口有四种工作方式——方式 0、方式 1、方式 2 和方式 3。

1．方式 0

当 SM0=0、SM1=0 时，串行口工作于方式 0（移位寄存器方式）。在方式 0 下，RXD（P3.0）为数据输入/输出端，TXD（P3.1）同步脉冲输出端。发送或接收的数据为 8 位，低位在前，高位在后。方式 0 的波特率固定为 $f_{osc}/12$，也就是每个机器周期传送一位数据。

（1）发送。当把要发送的数据写入串行口发送缓冲器 SBUF 后，串行口即将此 8 位数据

以 f_{osc}/12 的波特率从 RXD 引脚输出，低位在前，高位在后。发送完 8 位数据后，由硬件将 TI 标志置位。再次发送数据之前，必须由软件将 TI 清零。方式 0 发送时序如图 7.4 所示。

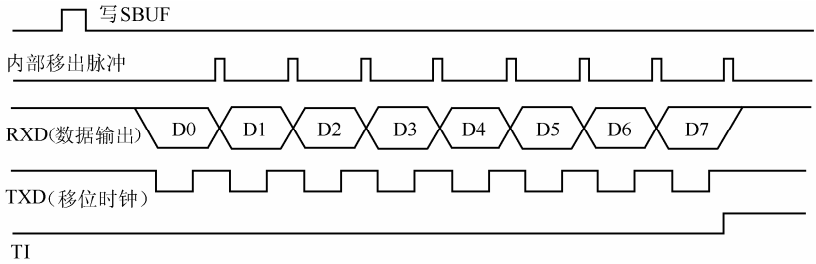


图 7.4　方式 0 发送时序

（2）接收。在满足 REN=1 和 TI=0 的条件下，就会启动一次接收过程，此时 RXD 为数据输入端，TXD 为同步信号输出端。串行输入的波特率为 f_{osc}/12。当接收完 8 位数据后，由硬件将 RI 标志置位。当再次接收时，必须由软件将 RI 标志清零。方式 0 接收时序如图 7.5 所示。

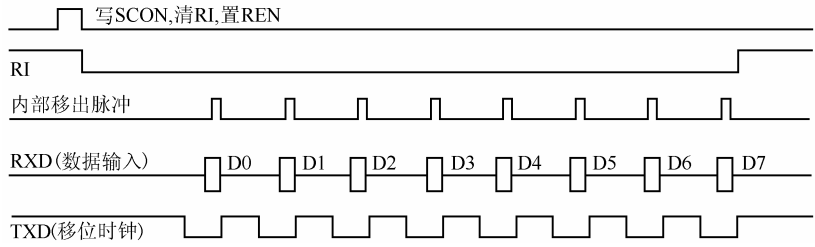


图 7.5　方式 0 接收时序

利用串行口方式 0，外接移位寄存器，可以扩展并行 I/O 接口，限于篇幅，在此不再叙述，请查阅相关书籍。

2．方式 1

当 SM0=0、SM1=1 时，串行口工作于方式 1。方式 1 为波特率可变的 8 位异步通信方式，由 TXD 发送，RXD 接收。一帧数据为 10 位：1 位起始位（低电平）、8 位数据位（低位在前）和 1 位停止位（高电平）。

（1）发送。当 CPU 执行一条将数据写入发送缓冲器 SBUF 的指令，就启动发送。当发送完一帧数据时，发送中断标志 TI 被置位。串行口方式 1 发送时序如图 7.6 所示。

（2）接收。当 REN=1 时，允许接收。接收器开始检测 RXD 引脚的信号，采样频率为波特率的 16 倍。当检测到 RXD 引脚上从“1”到“0”的跳变时，就启动接收器接收。先接收起始位，然后接收一帧的其余信息。如果接收不到有效的起始位，则重新检测负跳变。

为了保证接收数据的正确无误，对每一位数据的检测采用“测三取二”的方法。即把一个位周期分为 16 个状态，在第 7、8、9 状态检测 RXD 引脚上的电平，取其≥2 的相同值作为测得值。此检测方法是在每位数据的中间位置采样，这样即使接收端的波特率有些差别，也不致发生错码或漏码。

方式 1 接收时，必须满足以下两个条件：一是 RI=0，即没有中断请求或上一帧数据接收完成时发出的中断请求已被响应，SBUF 中存放的上一帧数据已被取走；二是 SM2=0 或接收

到的停止位为 1，则接收数据有效。将 8 位数据送入 SBUF，停止位送 RB8，RI 置位。如果上述两个条件不满足，则接收到的这一帧数据就会丢失，接收器重新开始检测 RXD 引脚上的负跳变。方式 1 接收时序如图 7.7 所示。

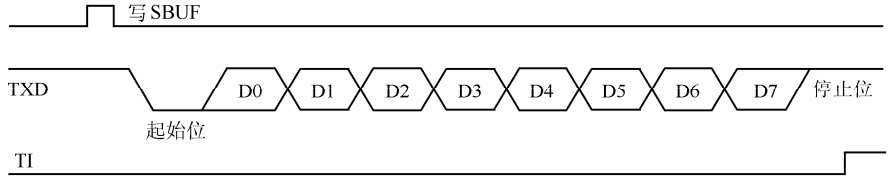


图 7.6　方式 1 发送时序

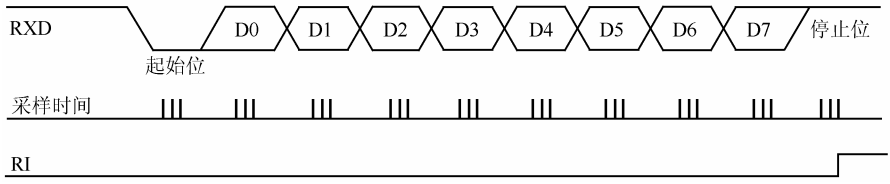


图 7.7　方式 1 接收时序

3．方式 2 和方式 3

当 SM0=1、SM1=0 时，串行口工作于方式 2，当 SM0=1、SM1=1 时，串行口工作于方式 3。这两种方式都是 9 位异步通信方式，适用于多机通信，它们的区别仅在于波特率不同。

在方式 2 或方式 3 下，数据由 TXD 发送，RXD 接收。一帧数据为 11 位：1 位起始位（低电平）、8 位数据位（低位在前）、1 位可编程位（第 9 位数据，用作奇偶校验或地址/数据选择）和 1 位停止位。与方式 1 相比，多了一位可编程位。发送时，第 9 位数据为 TB8，接收时，第 9 位数据送入 RB8。

（1）发送。当 CPU 执行一条写入发送缓冲器 SBUF 的指令后，便立即启动发送。发送的数据由 TXD 端输出，第 9 位数据为 SCON 中的 TB8。发送完一帧信息时，中断标志 TI 被置位。在发送下一帧信息之前，TI 必须由软件清零。第 9 位数据可作为数据的奇偶校验位，也可作为多机通信中的地址/数据信息标志位。

（2）当 REN=1 时，允许接收。接收器开始检测 RXD 引脚上的信号，检测和接收数据的方法与方式 1 相似。当接收器收到第 9 位数据后，若同时满足以下两个条件：一是 RI=0；二是接收到的第 9 位数据为 1，则接收数据有效，8 位数据装入 SBUF，第 9 位数据装入 RB8，并由硬件将 RI 置位。若不满足这两个条件，则接收到的这一帧数据将丢失，重新开始检测起始位。

方式 2 或方式 3 的发送与接收时序与方式 1 相似，仅仅是多了一位数据 TB8 或 RB8。

7.2.3　波特率的设置

MCS-51 单片机串行通信的波特率取决于串行口的工作方式。

方式 0 的波特率是固定的，为 $f_{osc}/12$。

方式 2 的波特率有两种，取决于 SMOD（PCON.7），SMOD=1 时为 $f_{osc}/32$，SMOD=0 时为 $f_{osc}/64$。由于 PCON 无位寻址功能，通常通过以下指令清零或置 1。

```
ANL   PCON，#7FH    ；使 SMOD=0
ORL   PCON，#80H    ；使 SMOD=1
```

当串行口工作于方式 1 和方式 3 时，波特率取决于定时器/计数器 1 的溢出率及 SMOD，并有如下关系式：

$$\text{波特率}=2^{\text{SMOD}}\times\text{T1 溢出率}/32$$

式中，T1 溢出率=1/T1 溢出周期，而溢出周期即为 T1 的定时时间。在应用时，往往根据所需的波特率，先选取 SMOD，计算出 T1 的溢出率或溢出周期，然后计算 T1 的时间常数。T1 可以工作在方式 0、方式 1 和方式 2。但由于方式 2 为自动装入时间常数的 8 位定时器，使用时只需进行初始化，不需要在中断服务程序中重装时间常数，因而是一种常用方式。

设 T1 工作于方式 2，定时器初值为 N，则有如下计算式：

$$\text{波特率}=2^{\text{SMOD}}\times\text{T1 溢出率}/32=2^{\text{SMOD}}\times f_{\text{osc}}/[32\times12(2^8-N)]$$

$$N=256-2^{\text{SMOD}}\times f_{\text{osc}}/(384\times\text{波特率})$$

例如，要求串行口以方式 1 工作，通信波特率为 2400b/s，设振荡频率 f_{osc} 为 6MHz，选 SMOD=1，则 T1 时间常数为：

$$N=256-2^1\times6\times10^6/(384\times2400)=242.98\approx243=\text{F3H}$$

定时器 T1 和串行口的初始化程序如下：

```
MOV TMOD，#20H    ；设置 T1 为方式 2
MOV TH1，#0F3H    ；置时间常数
MOV TL1，#0F3H
SETB TR1          ；启动 T1
ORL  PCON，#80H   ；SMOD=1
MOV SCON，#50H    ；设串行口为方式 1
```

为了应用方便，表 7.2 列出了常用波特率值和对应的晶振频率，以及在使用 T1 作为波特率产生器的工作模式和计数初值。

表 7.2　T1 产生的波特率及其计数初值

波特率	f_{osc}(MHz)	SMOD	T1			波特率	f_{osc}(MHz)	SMOD	T1		
			C/$\overline{\text{T}}$	模式	初值				C/$\overline{\text{T}}$	模式	初值
4800	16	1	0	2	EFH	2400	16	0	0	2	EFH
2400	16	1	0	2	DDH	1200	16	0	0	2	DDH
1200	16	1	0	2	BBH	600	16	0	0	2	BBH
600	16	1	0	2	75H	300	16	0	0	2	75H
4800	12	1	0	2	F3H	2400	12	0	0	2	F3H
2400	12	1	0	2	E6H	1200	12	0	0	2	E6H
1200	12	1	0	2	CCH	600	12	0	0	2	CCH
600	12	1	0	2	98H	300	12	0	0	2	98H
300	12	1	0	2	30H	110	12	0	0	1	FEEBH
56800	11.0592	1	0	2	FFH	9600	11.0592	0	0	2	FDH
19200	11.0592	1	0	2	FDH	4800	11.0592	0	0	2	FAH
9600	11.0592	1	0	2	FAH	2400	11.0592	0	0	2	F2H
4800	11.0592	1	0	2	F4H	1200	11.0592	0	0	2	E8H
2400	11.0592	1	0	2	E8H	600	11.0592	0	0	2	D0H
1200	11.0592	1	0	2	D0H	300	11.0592	0	0	2	A0H
600	11.0592	1	0	2	A0H	1200	6	0	0	2	F3H
300	11.0592	1	0	2	40H	110	6	0	0	2	72H

7.2.4 串行通信接口标准

常用的串行通信接口标准有 RS-232C、RS-422A 和 RS-485 等。

1．RS-232C通信接口

（1）接口信号。RS-232C 通信接口又称为 RS-232C 总线标准。它向外部的连接器有 25 针和 9 针两中“D”型插头，各针的功能及排列如图 7.8 和图 7.9 所示。

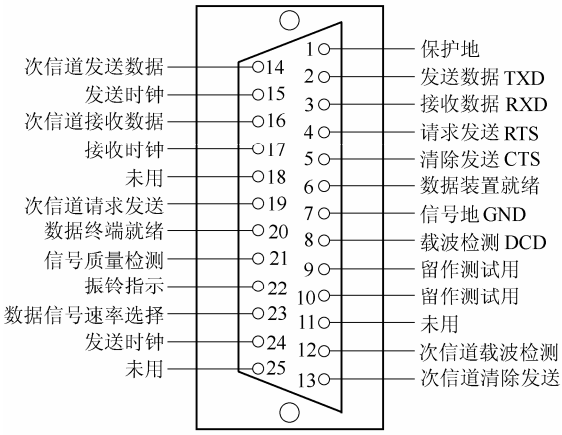


图 7.8　25 针插头引脚定义图

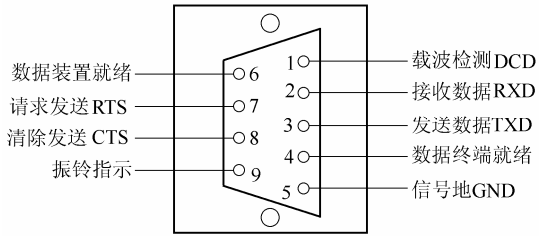


图 7.9　9 针插头引脚定义图

（2）电气特性。RS-232C 的电气特性如下：

逻辑“1”：−3V～−15V。

逻辑“0”：+3V～+15V。

传输距离：≤15 米。

最大负载电容：≤2500pF。

波特率：≤20kb/s。

接收器输入阻抗：3～7kΩ。

驱动器输出阻抗：≤300Ω。

驱动器转换速率：≤30V/μs。

输出短路电流：≤0.5A。

（3）电平转换。计算机电平通常是 TTL 电平，它与 RS-232C 电平不兼容，必须进行电平转换。

RS-232C 与 TTL 的电平转换的芯片有传输线驱动器 MC1488 和传输线接收器 MC1489。由于 MC1488 和 MC1489 需要±15V 或±12V 供电，使用不方便，现常用单+5V 电的转换芯片有 MAXIM 公司的 MAX232 芯片，它可以实现 RS-232C 与 TTL/CMOS 电平之间的转换。MAX232 的组成及引脚图如图 7.10 所示。

在实际应用中，MAX232 器件对电源噪声很敏感，因此电源 V_{CC} 应加上 1μF 的去耦电容，图中 C1、C2、C3、C4 取 1.0μF/16V 的钽电解电容，安装时尽量靠近器件，以提高抗干扰能力。

2．RS-422A通信接口

（1）接口信号。RS-422A 通信接口是对 RS-232C 通信接口的改进，它采用平衡传输电气

标准，输入/输出均采用差分驱动，因此具有更强的抗干扰能力，传送速率大大提高。它向外部的连接器常采用 9 针“D”型插头，各针的功能及排列如图 7.11 所示。

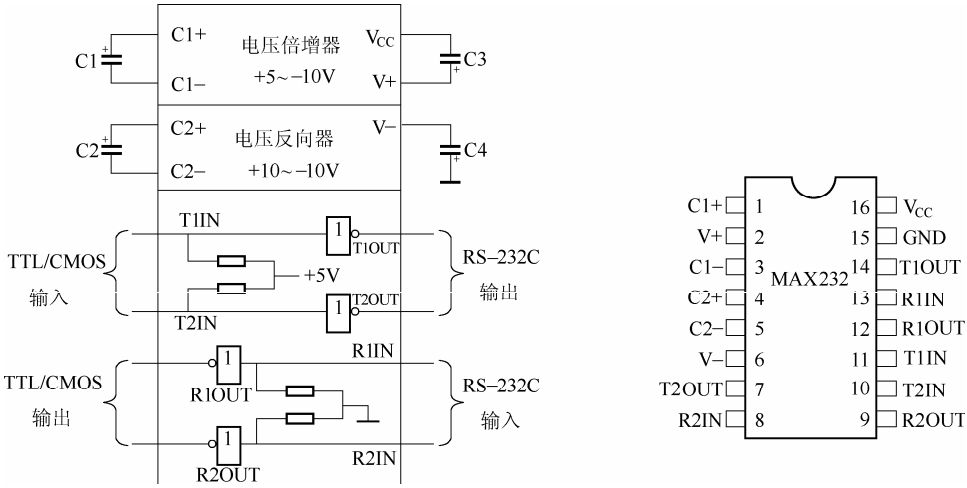


图 7.10　MAX232 组成及引脚图

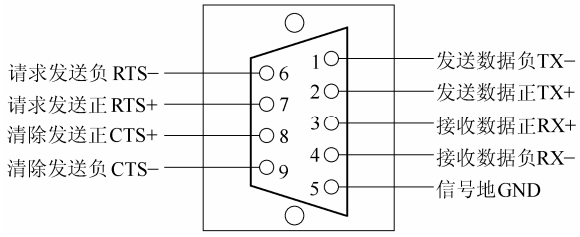


图 7.11　RS-422 插头引脚定义图

（2）电平转换。TTL 电平转换为 RS-422A 电平的常用芯片有 SN75174、MC3487 等。RS-422A 电平转换为 TTL 电平的常用芯片有 SN75175、MC3486 等。SN75174、SN75175 分别具有三态输出的单片四差分驱动器和接收器，符合 EIA 标准 RS-422A 规范，采用+5V 单电源供电，功能上分别可以与 MC3487、MC3486 互换。图 7.12 是用 SN75174、SN75175 实现的电平转换电路图。

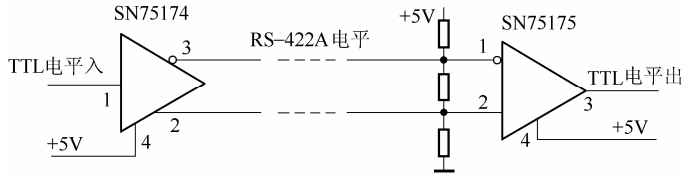


图 7.12　RS-422A 接口电平转换电路

3．RS-485 通信接口

RS-485 是 RS-422A 的变型，它与 RS-422A 的区别在于：RS-422A 为全双工，采用两对平衡差分信号线；而 RS-485 为半双工，采用一对平衡差分信号线。RS-485 对于多站互连是十分方便的。实现 TTL 电平与 RS-485 电平之间的转换可以采用专用的芯片，如 SN75176。也可以通过将 RS-422A 的发送数据信号正端与接收数据信号正端连起来，发送数据信号负端

与接收数据信号负端连起来实现。

4．RS232C、RS-422A/485 通信接口的性能比较

RS232C、RS-422A/485 通信接口的性能如表 7.3 所示。

表 7.3　RS-232C、RS-422A/485 通信接口的性能

接口 / 性能	RS-232C	RS-422A	RS-485
功能	双向，全双工	双向，全双工	双向，半双工
传输方式	单端	差分	差分
逻辑“0”电平	3V～15V	2V～6V	1.5V～6V
逻辑“1”电平	–3V～–15V	–2V～–6V	–1.5V～–6V
最大速率	20kb/s	10Mb/s	10Mb/s
最大距离	30m	1200m	1200m
驱动器加载输出电压	±5V～±15V	±2V	±1.5V
接受器输入敏感度	±3V	±0.2V	±0.2V
接收器输入阻抗	3～7KΩ	>4KΩ	>7KΩ
组态方式	点对点	1 台驱动器：10 台接收器	32 台驱动器：32 台接收器
抗干扰能力	弱	强	强
传输介质	扁平或多芯电缆	二对双绞线	一对双绞线

7.3　PC机与单片机通信

7.3.1　PC机串口资源及编程使用方法

PC 机具有功能十分完善的异步通信接口，其核心芯片为 INS8250 或与其兼容的芯片。其端口地址范围是 COM1：3F8H～3FFH，COM2：2F8H～2FFH。现在有些计算机还有 COM3：3E8H～3EFH，COM4：2E8H～2EFH。PC 机的串行口可以与调制解调器配合实现远距离通信，传输波特率为 50～9600。

PC 机串行口资源状况如表 7.4 所示。

表 7.4　PC 机串行口资源状况

寄存器名称	端口地址		复位后初始值
	COM1	COM2	
发送器保持寄存器（TBR）	3F8H	2F8H	××××××××B
接收器数据寄存器（RBR）	3F8H	2F8H	××××××××B
波特绿因子低位寄存器（DLL）	3F8H	2F8H	00000000B
波特绿因子高位寄存器（DLH）	3F9H	2F9H	00000001B
中断允许寄存器（IER）	3F9H	2F9H	00000000B
中断标识寄存器（IIR）	3FAH	2FAH	00000000B
线路控制寄存器（LCR）	3FBH	2FBH	01100000B
调制解调器控制寄存器（MCR）	3FCH	2FCH	××××0000B
线路状态寄存器（LSR）	3FDH	2FDH	××××××××B
调制解调器状态寄存器（MSR）	3FEH	2FEH	××××××××B

使用 PC 机串行口时，必须先进行初始化。初始化要涉及端口寄存器每一位的功能，请读者参阅 INS8250 的有关资料，主要有以下四方面的工作。

1．设置波特率

为了方便使用，表 7.5 列出波特率因子选择表。

表 7.5　波特率选择

DLH	DLL	波特率	DLH	DLL	波特率	DLH	DLL	波特率
09H	00H	50	00H	C0H	600	00H	18H	4800
06H	00H	75	00H	60H	1200	00H	10H	7200
04H	17H	110	00H	40H	1800	00H	0CH	9600
03H	59H	134.5	00H	3AH	2000	00H	06H	19200
03H	00H	150	00H	30H	2400	00H	03H	38400
01H	80H	300	00H	20H	3600	00H	01H	115200

假设波特率为 9600，以 COM1 为例，用 C 语言写出程序为：

```
outportb(0x3fb，0x80)；     //写线路控制寄存器，访问波特率因子寄存器
outportb(0x3f8，0x0c)；     //写入波特率因子低字节
outportb(0x3f9，0x00)；     //写入波特率因子高字节
```

2．设置数据格式

假设数据为 7 位，1 位起始位，1 位停止位，偶校验，则设置如下：

```
outportb(0x3fb，0x1a)；
```

3．设置操作方式

PC 机异步通信适配器中的 8250 中断输出（INTRPT）外接成受引脚 $\overline{\text{OUT2}}$ 控制输出的三态门控制。只有当 $\overline{\text{OUT2}}$ 为低时，并有 INTRPT 产生，中断信号才可通过三态门。因此只要控制 $\overline{\text{OUT2}}$ 输出，即可控制中断信号是否通过。

不允许中断：

```
outportb(0x3fc，0x03)；
```

允许中断：

```
outportb(0x3fc，0x0b)；
```

自测试工作方式，且不允许中断：

```
outportb(0x3fc，0x13)；
```

4．设置中断允许寄存器

计算机在复位后，中断处于关闭状态，可用下面语句打开：

```
outportb(0x3f9，0x0f)；
```

例 7.1　将 PC 机的 COM1 初始化为 9600 波特率、7 位数据、偶校验、1 位停止位、不中断，则初始程序如下：

```
initcom1()
{
```

```
    Outportb(0x3fb，0x80)；
    outportb(0x3f8，0x0c)；
    outportb(0x3f9，0x00)；
    outportb(0x3fb，0x1a)；
    outportb(0x3fc，0x03)；
}
```

例 7.2　PC 机从 COM1 发送一个字符，程序如下：

```
send(char sendchar)
{
  int coms;
  do{                                  //查询发送缓冲器是否空
      coms=inportb(0x3fd)&0x20；
    }while(coms!=0x20)；
  outportb(0x3f8，sendchar)；
}
```

例 7.3　PC 机从 COM1 接收一个字符，程序如下：

```
char receive()
{
  char coms，receivechar；
  waitok：                             //查询接收数据是否有效
      coms=inportb(0x3fd)；
      if((coms&0x01)==0)
      goto waitok；
      receivechar= inportb(0x3f8)；
  return receivechar；
}
```

7.3.2　PC机与单片机双机通信

在工程应用中，常常由一台 PC 机和一台单片机构成主从式计算机测控系统。在这样的系统中，以单片机为核心的智能测控仪表（从机）作为现场测控设备，完成数据的采集、处理和各种控制任务，同时将数据传送给 PC 机（主机），PC 机将这些数据进行加工处理后，进行显示、打印报表等，PC 机也可以将各种控制命令传送给单片机，干预单片机系统的运行，从而充分发挥 PC 机的优势。要实现这样的功能，就要解决 PC 机与单片机之间的数据通信问题。首先讨论 PC 机与一台单片机系统进行点对点的通信问题。

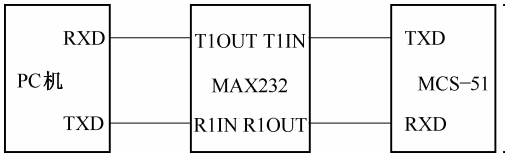


图 7.13　PC 机与单个单片机通信电路

如图 7.13 所示，PC 机与 MCS-51 单片机通过 RS-232C 总线相连。为了弄清通信软件的

设计，只举一个简单的通信例子：PC 机从键盘上输入一个字符，然后将这个字符发送给单片机，单片机接收到这个字符后，不做任何处理，又将它发回给 PC 机，PC 机将这个字符显示在屏幕上，如果显示的字符与输入的字符一致，则表明 PC 机与单片机之间的通信正常。

为了实现通信，双方约定如下：

波特率：2400b/s。

信息格式：8 位数据，1 位停止位。

传送方式：PC 机采用查询式收发数据，单片机采用中断式收发数据。

1．PC机的通信软件

通信软件采用 Turbo C 编写，程序流程图如图 7.14 所示。程序如下：

```
#include “stdio.h”
main()
{
  char c1，c2；
  int coms；
  outportb(0x3fb，0x80)；          //初始化 8250
  outportb(0x3f8，0x30)；
  outportb(0x3f9，0x00)；
  outportb(0x3fb，0x03)；          //8 位数据，1 位停止位
  outportb(0x3fc，0x03)；
  c1=getchar()；
  do{                              //查询发送缓冲器是否空
      coms=inportb(0x3fd)&0x20；
    }while(coms!=0x20)；
  outportb(0x3f8，c1)；
  waitok：                         //查询接收数据是否有效
    coms=inportb(0x3fd)；
    if((coms&0x01)==0)
  goto waitok；
  c2=inportb(0x3f8)；
  printf("The return char is：")；
  putchar(c2)；
}
```

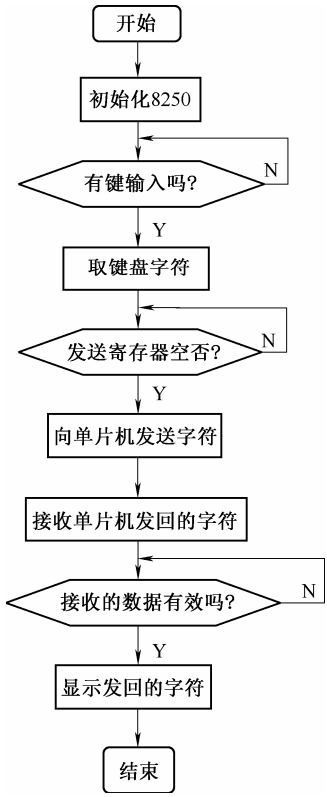


图 7.14　双机通信时 PC 机程序流程图

2．MCS-51 单片机通信软件

MCS-51 单片机采用中断方式接收 PC 机发过来的字符，并回送给主机。程序约定如下：

由于波特率为 2400b/s，设 T1 工作于方式 2，SMOD=1，系统时钟为 6MHz ，所以计数常数为 F3H。

单片机串行口工作于方式 1，允许接收。程序流程图如图 7.15 所示。

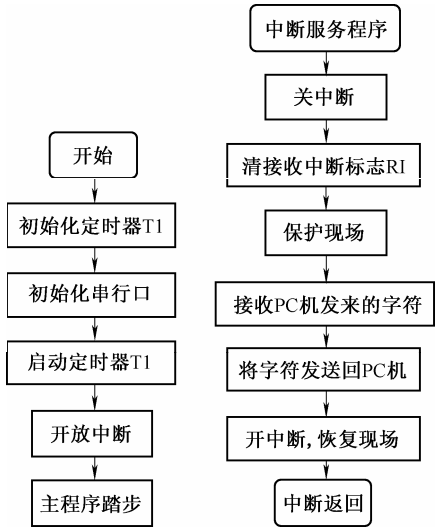


图 7.15　双机通信时 MCS-51 单片机程序流程图

程序如下：

```
        ORG    0000H
        LJMP   MAIN
        ORG    0023H
        LJMP   SERVE
MAIN:   MOV    TMOD，#20H      ；初始化 T1
        MOV    TH1，#0F3H
        MOV    TL1，#0F3H
        MOV    SCON，#50H      ；串行口为方式 1，REN=1
        MOV    PCON，#80H      ；SMOD=1
        SETB   TR1
        SETB   EA
        SETB   ES
HERE:   SJMP   HERE
SERVE:  CLR    EA              ；中断服务程序
        CLR    RI
        PUSH   PSW             ；保护现场
        SETB   RS0
        CLR    RS1
        PUSH   ACC
        MOV    A，SBUF         ；接收 PC 机发来的字符
        MOV    SBUF，A         ；将字符回送给 PC 机
WAIT:   JNB    TI，WAIT        ；等待发送完毕
        CLR    TI
        POP    ACC             ；恢复现场
        POP    PSW
        SETB   EA
```

```
        RETI
        END
```

7.3.3 PC机与单片机多机通信

PC 机不仅可以下挂一台单片机，而且可以下挂多台单片机实现多机通信。PC 机通常作为主机，而单片机通常作为从机。在此就存在一个问题是：PC 机如何确定与多台单片机中哪一台进行通信。通常的做法是把单片机进行编号，这个编号叫做地址或站号，当 PC 机需要同某一单片机通信时，首先发出单片机的编号，每台单片机同时收到编号，并与自己的编号比较，当相同时，就把编号反馈给 PC 机，从而建立起通信通路。我们已经知道，MCS-51 单片机的串行口有四种通信工作方式，其中方式 2 和方式 3 是专为多机通信而设置的。在方式 2 和方式 3 中，用户通过使用多机通信控制位 SM2，可以实现主机与从机的通信。

PC 机的串行口是以 8250 为核心部件组成的。虽然 8250 本身不具备多机通信功能，但可通过软件的办法，使得 8250 满足 MCS-51 单片机多机通信的要求。具体方法如下：

8250 可发送 11 位数据帧，这 11 位数据帧由 1 位起始位、8 位数据位、1 位奇偶校验位和 1 位停止位组成，其格式如下：

起始位	D0	D1	D2	D3	D4	D5	D6	D7	奇偶位	停止位

而 MCS-51 单片机多机通信的数据帧格式为：

起始位	D0	D1	D2	D3	D4	D5	D6	D7	TB8	停止位

其中 TB8 是可编程的，通过使其为 0 或 1，将数据帧和地址帧区别开来。

比较上述两种数据格式可知，它们的数据长度相同，不同的仅在于奇偶校验位和 TB8。如果我们通过软件的方法，编程 8250 的奇偶校验位，使得在发送地址时为 1，发送数据时为 0，则 8250 的奇偶校验位完全模拟了单片机多机通信的 TB8 位，对于这一点是不难办到的，只要给 8250 的线路控制寄存器写入特定的控制字即可。

使 8250 发送帧的奇偶校验位为 1 的语句是（以 COM1 为例）：

```
outportb(0x3fb，0x2b)；
```

PC 机在发送地址前，可执行上述语句。

使 8250 发送帧的奇偶校验位为 0 的语句是（以 COM1 为例）：

```
outportb(0x3fb，0x3b)；
```

PC 机在发送数据前，可执行上述语句。

如图 7.16 所示的多机通信电路，从 PC 机的键盘上输入单片机的编号，要求 PC 机与该单片机进行通信测试，从键盘上输入一个字符，发送给某单片机，单片机收到该字符后，又发回给 PC 机，在屏幕上显示出来。

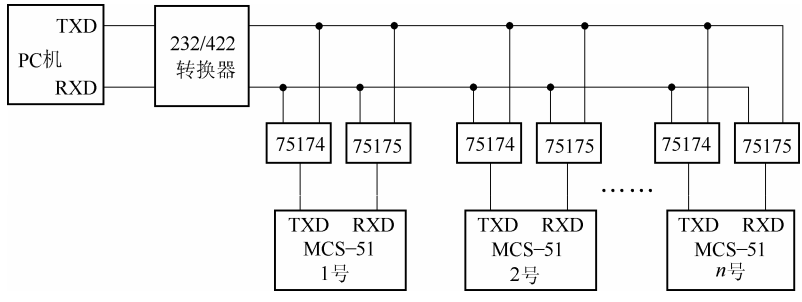


图 7.16　PC 机与多个单片机通信电路

1．PC机软件设计

PC 机采用查询方式，程序流程图如图 7.17 所示。程序如下：

```
#include "stdio.h"
main()
{
  char c1，c2，n1，n2；
  int coms；
  outportb(0x3fb，0x80)；          //初始化 8250
  outportb(0x3f8，0x30)；
  outportb(0x3f9，0x00)；
  outportb(0x3fb，0x2b)；       //8 位数据，1 位停止，
                                奇偶校验位为 1
  outportb(0x3fc，0x03)；     //PC 机处于发送地址状态
  outportb(0x3f9，0x00)；     //禁止 8250 中断
  printf("Please input the number of MCS：\n")；
  scanf("%c"，&n1)；
  do{                          //查询发送移位寄存器是否空
      coms=inportb(0x3fd)&0x40；
      }while(coms!=0x40)；
  outportb(0x3f8，n1)；//PC 机发送单片机地址编号
  outportb(0x3fc，0x01)；          //PC 机处于接收状态
  waitok1：           //查询单片机返回的编号是否有效
      coms=inportb(0x3fd)；
      if((coms&0x01)==0)
  goto waitok1；
  n2=inportb(0x3f8)；          //读入单片机返回的编号
  if(n1!=n2)
      printf("The number of MCS is error!\n")；
      break；
  else
      {
      outportb(0x3fc，0x03)；     //PC 机处于发送状态
      outportb(0x3fb，0x3b)；     //PC 机处于发送数据状态
      c1=getchar()；
      do{                          //查询发送移位寄存器是否空
          coms=inportb(0x3fd)&0x40；
      }while(coms!=0x40)；
      outportb(0x3f8，c1)；
      outportb(0x3fc，0x01)；              //PC 机处于接收状态
  waitok2：                                //查询单片机返回的字符是否有效
      coms=inportb(0x3fd)；
```

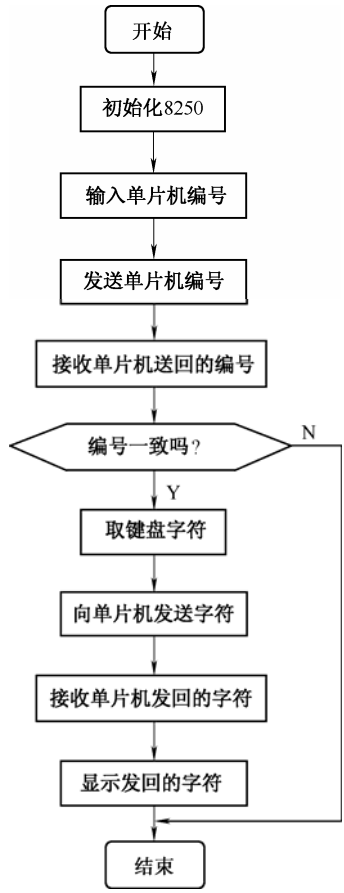


图 7.17　多机通信时 PC 机程序流程图

```
        if((coms&0x01)==0)
        goto waitok2;
        c2=inportb(0x3f8);                    //读入单片机返回的字符
        printf("The return char is：");       //显示返回的字符
        putchar(c2);
    }
}
```

2．单片机软件设计

单片机作为从机，串行通信采用中断方式。初始化后，处于接收状态。设系统时钟为6MHz。主程序与双机通信时是一样的，中断服务程序的流程图如图 7.18 所示。

假设单片机是 1 号机，程序如下：

```
        ORG     0000H
        LJMP    MAIN
        ORG     0023H
        LJMP    SERVE
MAIN：  MOV     TMOD，#20H      ；初始化 T1
        MOV     TH1，#0F3H
        MOV     TL1，#0F3H
        MOV     SCON，#0F0H     ；串行口为方式 3，REN=1
        MOV     PCON，#80H      ；SMOD=1
        SETB    TR1
        SETB    EA
        SETB    ES
HERE：  SJMP    HERE
SERVE： CLR     EA
        PUSH    ACC
        PUSH    PSW
        CLR     RS1
        SETB    RS0
        MOV     A，SBUF         ；接收地址
        XRL     A，#01H         ；与本机地址进行比较
        JNZ     RETURN          ；与本机地址不符，则返回
        CLR     SM2             ；置单片机为接收数据状态
        CLR     RI
        MOV     A，#01H
        MOV     SBUF，A         ；发送本机地址给 PC 机
        JNB     TI，$           ；等待发送完毕
        CLR     TI
        JNB     RI，$           ；等待 PC 机发来的字符接收完毕
```

```
        MOV    A，SBUF    ；读入字符
        CLR    RI
        MOV    SBUF，A
        JNB    TI，$      ；等待发送完毕
        CLR    TI
RETURN：POP    PSW
        POP    ACC
        SETB   EA
        RETI
        END
```

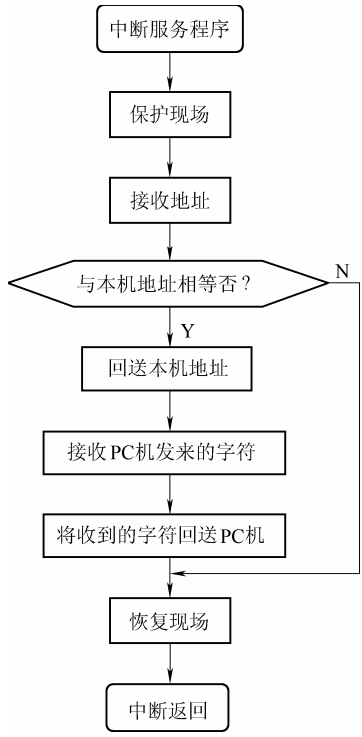


图 7.18　多机通信是单片机的中断服务程序流程图

本 章 小 结

通信是信息社会的普遍要求，单片机作为最前沿的自动控制设备或智能仪器的核心芯片，它往往是整个计算机网络中的一个环节或一个点，因此串行通信是比较重要的内容，具有很强的现实意义。本章要求：了解通信、并行通信、串行通信、同步通信、异步通信、信息帧、波特率等概念；了解 MCS-51 单片机串行口的结构；掌握 MCS-51 单片机的 4 种工作方式及波特率的设置；掌握 4 种方式下串行数据的收发过程；了解 RS232C、RS422A/485 通信协议；了解 PC 机串口资源及其使用方法；掌握单片机与 PC 机进行双机和多机通信的电路及实现程序。

习　题　7

7.1　MCS-51 单片机串行口有几种工作方式？各种方式的帧格式如何？波特率又如何确定？

7.2　以 MCS-51 单片机串行口方式 1 为例，说明数据的发送和接收过程。

7.3　MCS-51 中 SCON 的 SM2、TB8、RB8 有何作用？

7.4　若晶振频率为 11.0592MHz，串行口工作于方式 1，波特率为 4800b/s，T1 工作于方式 2 作为波特率发生器，试计算 T1 的时间常数，并编写初始化程序。

7.5　常用的串行通信总线有哪些？各有什么特点？

7.6　为什么说 RS-485 通信总线是工程中应用普遍的一种串行通信总线？

7.7　PC 机与单片机的多机通信是如何实现的？

7.8　设计一个 PC 机与单片机的双机通信系统，PC 机要求单片机将从 30H 开始的 8 个单元的数据送给 PC 机。试设计电路和相应的通信程序。

第 8 章　C51 程序设计语言及程序设计

本章主要内容

C51 数据类型。C51 流程控制语句。C51 函数。C51 应用编程实例。

近年来，设计人员在单片机的开发应用中，越来越多地使用 C 语言，因为 C 语言具有容易阅读和维护，可移植性好的特点，从而大大减少了开发时间。对于大多数 51 系列单片机，使用 C 语言这样的高级语言与使用汇编语言相比具有如下优点：

（1）不需要了解处理器的指令集，也不必了解存储器结构。

（2）寄存器分配和寻址方式由编译器进行管理，编程时不需要考虑存储器的寻址和数据类型等细节。

（3）C 语言中提供丰富的库函数，用户根据需要调用，大大缩短了程序开发和调试时间。

（4）易实现模块化设计。

（5）C 语言可移植性好，几乎适用于所有的目标系统，功能化的代码能够较方便地从一个系统移植到另一个系统。

C 语言是一门应用非常普遍的高级程序设计语言，因此本书并不主要介绍 C 语言的基本知识，而是主要分析 51 系列单片机 C 语言（以下简称 C51）。

8.1　C51 数据类型与运算

8.1.1　C51 数据类型

C51 具有标准 C 语言的所有标准数据类型，针对 51 单片机内部结构增加了以下特殊数据类型：

（1）bit　位变量。

（2）sbit 可独立寻址的位变量。

（3）sfr　8 位特殊功能寄存器。

（4）sfr16　16 位特殊功能寄存器。

Keil uVision2 是美国 Keil Software 公司出品的 51 系列兼容单片机 C 语言软件开发系统，表 8.1 列出了 Keil uVision2 单片机 C 语言编译器所支持的数据类型。在标准 C 语言中基本的数据类型为 char，int，short，long，float 和 double，而在 C51 编译器中 int 和 short 相同，float 和 double 相同，这里就不列出说明了。

1．char字符类型

char 类型的数据长度是一个字节，通常用于定义处理字符数据的变量或常量。分无符号字符类型 unsigned char 和有符号字符类型 signed char，默认值为 signed char 类型。unsigned char 类型用字节中所有的位来表示数值，所能表达的数值范围是 0～255。 signed char 类型

用字节中最高位字节表示数据的符号，“0”表示正数，“1”表示负数，负数用补码表示。所能表示的数值范围是−128～+127。unsigned char 常用于处理 ASCII 字符或用于处理小于或等于 255 的整型数。

表 8.1　Keil uVision2 单片机 C 语言编译器所支持的数据类型

数 据 类 型	位数（bit）	字节数（byte）	值 域 范 围
unsigned char	8	1	0～255
signed char	8	1	−128～127
unsigned int	16	2	0～65535
signed int	16	2	−32768～32767
unsigned long	32	4	0～4294967295
signed long	32	4	−2147483648～+2147483647
float	32	4	±1.175494E-38～±3.402823E+38
一般指针	24	3	存储空间 0～65535
bit	1	—	0、1
sbit	1	—	0、1
sfr	8	1	0～255
sfr 16	16	2	0～65535

2．int整型

int 整型长度为两个字节，用于存放一个双字节数据。分有符号整型数 signed int 和无符号整型数 unsigned int，默认值为 signed int 类型。signed int 表示的数值范围是−32768～+32767，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数。unsigned int 表示的数值范围是 0～65535。

3．long长整型

long 长整型长度为四个字节，用于存放一个四字节数据。分有符号长整型 signed long 和无符号长整型 unsigned long，默认值为 signed long 类型。signed long 表示的数值范围是−2147483648～+2147483647，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数。unsigned long 表示的数值范围是 0～4294967295。

4．float浮点型

float 浮点型在十进制中具有 7 位有效数字，是符合 IEEE−754 标准的单精度浮点型数据，占用四个字节。

5．指针型

指针型本身就是一个变量，在这个变量中存放的数据是指向另一个数据的地址。这个指针变量要占据一定的内存单元，对不一样的处理器长度也不尽相同，在 C51 中它的长度一般为 1～3 个字节。

6．bit位变量

bit 位变量是 C51 编译器的一种扩充数据类型，利用它可定义一个位标量，但不能定义位

指针，也不能定义位数组。它的值是一个二进制位，不是 0 就是 1，类似一些高级语言中的 Boolean 类型中的 True 和 False。

7．sfr特殊功能寄存器

sfr也是一种扩充数据类型，占用一个内存单元，值域为 0～255。利用它能访问 51单片机内部的所有特殊功能寄存器。

8．sfr16 16 位特殊功能寄存器

sfr16 占用两个内存单元，值域为 0～65535。sfr16 和 sfr 一样用于操作特殊功能寄存器，不同的是它用于操作占两个字节的寄存器，如定时器 T2。

9．sbit 可寻址位

sbit 同样是单片机 C 语言中的一种扩充数据类型，利用它能访问芯片内部的 RAM 中的可寻址位或特殊功能寄存器中的可寻址位。

8.1.2 C51 数据存储类型

C51 编译器可以通过将变量、常量定义为不同的存储类型（data，bdata，idata，pdata，xdata，code）的方法，将它们定义在不同的存储区中。

C51 数据存储类型与 MCS-51 单片机实际存储空间的对应关系如表 8.2 所示。

表 8.2 C51 存储类型与 MCS-51 单片机存储空间的对应关系

存 储 类 型	与存储空间的对应关系
data	直接寻址片内数据存储区，访问速度快（128 字节）
bdata	可位寻址片内数据存储区，允许位与字节混合访问（16 字节）
idata	间接寻址片内数据存储区，可访问片内全部 RAM 地址空间（256 字节）
pdata	分页寻址片外数据存储区（256 字节），由 MOVX @Ri 访问
xdata	寻址片外数据存储区（64K 字节），有 MOVX @DPTR 访问
code	寻址代码存储区（64K 字节），有 MOVC @DPTR 访问

当使用存储类型 data，bdata 定义常量和变量时，C51 编译器会将它们定位在片内数据存储区中。片内 RAM 是存放临时性传递变量或使用频率较高变量的理想场所。访问片内数据存储器（data，bdata，idata）比访问片外数据存储器（xdata，pdata）相对快一些，因此可将经常使用的变量置于片内数据存储器，而将规模较大的，或不常使用的数据置于片外数据存储器中。

C51 存储类型及其大小和值域如表 8.3 所示。

表 8.3 C51 存储类型及其大小和值域

存 储 类 型	位数（bit）	字节（Byte）	值域范围
data	8	1	0～255
idata	8	1	0～255
pdata	8	1	0～255
xdata	16	2	0～65535
code	16	2	0～65535

例如，

```
char data var1;                          //字符变量 char var1 被定义为 data 存储类型，定位在片内 RAM 中
bit bdata flags;                         //位变量 flags 被定义为 bdata 存储类型，定位在片内 RAM 中的位寻址区（20H～2FH）
float idata x，y，z;                     //浮点变量 x，y，z 被定义为 idata 存储类型，定位在片内 RAM 中，并只能用间接寻址的方法进行访问
unsigned int pdata dimension;            //无符号整型变量 dimension 被定义为 pdata 存储类型，定位在片外数据存储区，并用 MOVX @Ri 访问
unsigned char xdata vector[10][4][4];    //无符号字符三维数组变量 unsigned char vector[10][4][4]被定义为 xdata 存储类型，定位在片外 RAM 中，占据 10×4×4=160 个字节空间
unsigned char code a[]={0x00,0x01};      //数组 a[]被定义为 code 存储类型，定位在程序存储器中
```

如果在变量定义时略去存储类型标志符，编译器会自动默认存储类型。默认的存储类型进一步有 SMALL、COMPACT 和 LARGE 存储模式指令限制。见表 8.4。

表 8.4　存储模式及说明

存储模式	说　　明
SMALL	参数及局部变量放入可直接寻址的片内存储器（最大 128 字节，默认存储类型是 data），因此访问十分方便。另外所有对象，包括栈，都必须嵌入片内 RAM。栈长很关键，因为实际栈长依赖于不同函数的嵌套层数
COMPACT	参数及局部变量放入分页片外存储区（最大 256 字节，默认的存储类型是 pdata），通过寄存器 R0 和 R1 间接寻址，栈空间位于内部数据存储区中
LARGE	参数及局部变量直接放入片外数据存储区（最大 64KB，默认存储类型为 xdata），使用数据指针 DPTR 来进行寻址。用此数据指针访问的效率较低，尤其是对二个或多个字节的变量，这种数据类型的访问机制直接影响代码的长度，另一不方便之处在于这种数据指针不能对称操作

存储模式决定了变量的默认存储类型，参数传递区和无明确存储类型的说明。例如，char var1 在 SMALL 存储模式下，var1 被定位在 data 存储区；在 COMPACT 模式下，var1 被定位在 idata 存储区；在 LARGE 模式下，var1 被定位在 xdata 存储区中。

8.1.3　C51 定义SFR

MCS-51 单片机内有 21 个特殊功能寄存器（SFR)，它们分布在片内 RAM 的高 128 字节中，对特殊功能寄存器只能用直接寻址方式访问。特殊功能寄存器中还有 11 个可位寻址的寄存器。

在 C51 中，特殊功能寄存器及其可位寻址的位是通过关键字 sfr 和 sbit 来定义的，这种方法与标准 C 不兼容，只适用于 C51。

（1）sfr：定义为 8 位特殊功能寄存器。例如，

```
sfr PSW=0xD0;          //定义程序状态字 PSW 的地址为 D0H
sfr TMOD=0x89;         //定义定时器/计数器方式控制寄存器 TMOD 的地址为 89H
sfr P1=0x90;           //定义 P1 口的地址为 90H
```

“sfr”后面必须跟一个特殊寄存器名；“=”后面的地址必须为常数，常数值的范围必须在特殊功能寄存器地址范围内，即位于地址 0x80 到 0xff 之间，不允许带有运算符的表达式。

（2）sfr16：定义的 16 位特殊功能寄存器。例如，

```
sfr16 T2=0xCC;              //定义 8052 定时器 2，低 8 位地址为 T2L=CCH，高 8 位 T2H=CDH
```

用 sfr16 定义 16 位特殊功能寄存器时，等号后面是它的低位地址。sfr16 不能用于定时器 0 和 1 的定义。

（3）sbit：定义为可位寻址对象，如访问特殊功能寄存器中的某位。例如，PSW 是可位寻址的 SFR，其中各位可用 sbit 定义。

```
sbit CY=0xD7;               //定义进位标志 CY 的地址为 D7H
sbit AC=0xD0^6;             //定义辅助进位标志 AC 的地址为 D6H
sbit RS0=0xD0^3;            //定义 RS0 的地址为 D3H
```

注意：sfr 和 sbit 只能在函数外使用，一般放在程序的开头。

实际上大部分特殊功能寄存器及其可位寻址的位的定义在 reg51.h、reg52.h 等头文件中已经给出，使用时只需在源文件中包含相应的头文件，即可使用 SFR 及其可位寻址的位；而对于未定义的位，使用前必须先定义。例如，

```
#include "reg51.h"
sbit P10=P1^0;
sbit P12=P1^2;
main()
{
  P10=1;
  P12=0;
  PSW=0x08;
  ……
}
```

8.1.4 C51 定义并行口

MCS-51 单片机的基本 I/O 口只有 P0、P1、P2、和 P3 四个，除此之外，还可以在片外扩展 I/O 口和其他功能芯片，它们与外部数据存储器统一编址，即 MCS-51 单片机把它们当作外部数据存储单元。

P0、P1、P2 和 P3 的定义在头文件 reg51.h 和 reg52.h 中，扩展的外部 RAM 单元和外部 I/O 口需要用户自己定义。例如，

```
#include "absacc.h"
#define PA XBYTE[0xffec]
main()
{
  PA=0x3A;                  //将数据 3AH 写入地址为 0xffec 的存储单元或 I/O 端口
}
```

以上程序用 C 中的编译预处理命令#define 将 PA 定义为外部 I/O 口，地址为 0xffec，是单字节量。其中 XBYTE 是一个指针，指向外部数据存储器的零地址单元，它是在绝对地址访问头文件 absacc.h 中定义的。

8.1.5 C51 定义位变量

MCS-51 单片机具有位运算器，C51 相应地设置了位数据类型。

1．位变量的定义

位变量用关键字“bit”来定义，它的值是一个二进制位。例如，

```
bit lock;                    //将 lock 定义为位变量
bit direction;               //将 direction 定义为位变量
```

2．函数可以有bit类型的参数，也可以有bit类型的返回值

例如，

```
bit func(bit b0，bit b1)
{
  bit a;
  ……
  return a;
  }
```

使用禁止中断宏命令#progma disable，或指定明确的寄存器切换（using n)的函数不能返回位值。

3．对位变量定义的限制

不能定义位变量的指针，如：bit *bit_point;

不能定义位数组，如：bit bit_array[5];

位变量说明中可以指定存储类型，位变量的存储类型只能是 bdata。

在程序设计时，对于可位寻址的对象，既可以字节寻址又可以位寻址的变量，则其存储类型只能是 bdata。

使用时，先说明字节变量的数据类型和存储类型。例如，

```
int bdata a;              //整型变量 a 定位在片内数据存储区中的可位寻址区
char bdata b[4];          //字符数组 b 定位在片内数据存储区中的可位寻址区
```

然后，使用 sbit 关键字定义其中可独立寻址的位变量。例如，

```
sbit a0=a^0;              //定义 a0 为 a 的第 0 位
sbit a12=a^12;            //定义 a12 为 a 的第 12 位
sbit b03=b[0]^3;          //定义 b03 为 b[0]的第 3 位
sbit b36=b[3]^6;          //定义 b36 为 b[3]的第 6 位
```

sbit 定义要求基址对象的存储类型为 bdata。

使用 sbit 类型位变量时，基址变量和其对应的位变量的说明必须在函数外部进行。

8.1.6 C51 运算符、表达式及其规则

C51 的运算符主要有：算术运算符、关系运算符、逻辑运算符、赋值及复合赋值运算符等。

1．算术运算符和算术表达式

（1）基本的算术运算符：C51 最基本的算术运算符有以下五种：

+ （加法运算符）

– （减法运算符）

＊（乘法运算符）

／（除法运算符）

% （模运算或取余运算符）

对于除法运算符：若两个整数相除，结果为整数（即取整)。

对于取余运算符：要求%两侧的操作数均为整型数据，所得结果的符号与左侧操作数的符号相同。

（2）自增、自减运算符：++为自增运算符，——为自减运算符。例如，

++j、j++、—i、i——

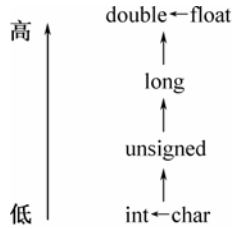
++和——运算符只能用于变量，不能用于常量和表达式。++j 表示先加 1，再取值；j++表示先取值，再加 1。同理，自减运算也是这个道理。

（3）算术表达式和运算符的优先级与结合性：用算术运算符和括号将运算对象连接起来的式子称为算术表达式。其中的运算对象包括常量、变量、函数、数组、结构等。例如：a+b*c/d。

C51 规定了算术运算符的优先级和结合性为：先乘除模，后加减，括号最优先。

如果一个运算符两侧的数据类型不同，则必须通过数据类型转换将数据转换成同种类型。转换方式有以下两种。

一是自动类型转换：即在程序编译时，由 C 编译器自动进行数据类型转换。转换规则如下：



一般来说，当运算对象的数据类型不相同时，先将较低的数据类型转换成较高的数据类型，运算结果为较高的数据类型。

二是强制类型转换：使用强制类型转换运算符，其形式为：（类型名）（表达式）。例如，

(double) a　将 a 强制转换成 double 类型

(int) (x+y)　将 x+y 强制转换成 int 类型

2．关系运算符和关系表达式

（1）关系运算符及其优先级：关系运算即比较运算。C51 提供了六种关系运算符如下：

＜　（小于）

＜=（小于等于）

＞　（大于）

＞=（大于等于）

==（等于）

！=（不等于）

优先级关系是：

①＜、＜=、＞、＞=这四个运算符的优先级相同，处于高优先级。

② ==和！=这两个运算符的优先级相同，处于低优先级。

关系运算符的优先级低于算术运算符的优先级，而高于赋值运算符的优先级。

（2）关系表达式：用关系运算符将运算对象连接起来的式子称为关系表达式。如：a>b，a+b>=c+d，(a=3)<(b=2)等。

关系表达式的值为逻辑值，即真和假。C51 中用 0 表示假，用 1 表示真。

例如，有关系表达式 a>=b，若 a 的值是 4，b 的值是 3，则给定关系满足，关系表达式的值为 1，即逻辑真；若 a 的值是 2，则给定关系不成立，系表达式的值为 0，即逻辑假。

3．逻辑运算符和逻辑表达式

（1）逻辑运算符及其优先级：逻辑运算是对逻辑量进行运算。C51 提供三种逻辑运算符。如下：

&&　（逻辑与）

||　（逻辑或）

！　（逻辑非）

它们的优先级关系是：！的优先级最高，而且高于算术运算符；||的优先级最低，它低于关系运算符，却高于赋值运算符。

（2）逻辑表达式：用逻辑运算符将运算对象连接起来的式子称为逻辑表达式。运算对象可以是表达式或逻辑量，而表达式可以是算术表达式、关系表达式或逻辑表达式。逻辑表达式的值也是逻辑量，即真或假。

对于算术表达式，其值若为 0，则认为是逻辑假；若不为 0，则认为是逻辑真。

逻辑表达式的执行规则是：逻辑表达式不一定完全被执行，只有当一定要执行下一个逻辑运算符才能确定表达式的值时，才执行该运算符。例如，

a&&b&&c

若 a 的值为 0，则不需判断 b 和 c 的值就可确定表达式的值为 0。

又如，a||b||c

若 a 值为 0，则还需判断 b 的值，若 b 的值为 1，则不需判断 c 的值就可确定表达式的值为 1。

4．位运算符及其表达式

位运算的操作对象只能是整型和字符型数据，不能是实型数据。C51 提供以下六种位运算。

&	（按位与）	相当于 ANL 指令
|	（按位或）	相当于 ORL 指令
^	（按位异或）	相当于 XRL 指令
～	（按位取反）	相当于 CPL 指令
<<	（左移）	相当于 RL 指令
>>	（右移）	相当于 RR 指令

5．赋值运算符和赋值表达式

（1）赋值运算符：赋值运算符就是赋值符号“=”，赋值运算符的优先级低，结合性是右结合性。

（2）赋值表达式：将一个变量与表达式用赋值号连接起来就构成赋值表达式。形式如下：

变量名=表达式

赋值表达式中表达式包括变量、算术运算表达式、关系运算表达式、逻辑运算表达式等，甚至可以是另一个赋值表达式。赋值过程是将“=”右边表达式的值赋给“=”左边的一个变量，赋值表达式的值就是被赋值变量的值。例如，

a=b=5，该表达式的值为 5。

a=(b=4)+(c=6)，该表达式的值为 10。

（3）赋值的类型转换规则：在赋值运算中，当“=”两侧的类型不一致时，系统自动将右边表达式的值转换成左侧变量的类型，再赋给该变量。转换规则如下：

① 实型数据赋给整型变量时，舍弃小数部分。

② 整型数据赋给实型变量时，数值不变，但以浮点数形式存储在变量中。

③ 长字节整型数据赋给短字节整型变量时，实行截断处理。如将 long 型数据赋给 int 型变量时，将 long 型数据的低两字节数据赋给 int 型变量，而将 long 型数据的高两字节的数据丢弃。

④ 短字节整型数据赋给长字节整型变量时，进行符号扩展。如将 int 型数据赋给 long 型变量时，将 int 型数据赋给 long 型变量的低两字节，而将 long 型变量的高两字节的每一位都设为 int 型数据的符号值。

6．复合赋值运算符

赋值号前加上其他运算符构成复合运算符。C51 提供以下十种复合运算符：

+=，−=，*=，/=，%=，&=，|=，^=，<<=，>>=。

例如，

a+=b	等价于 a=(a+b)
x*=a+b	等价于 x=(x*(a+b))
a&=b	等价于 a=(a&b)
a<<=4	等价于 a=(a<<4)

8.2 C51 流程控制语句

C51 程序与其他语言程序一样，程序结构也是分为顺序结构、选择结构或分支结构、循环结构三种。由于顺序结构比较简单，在此不再讲述，下面就选择语句和循环语句进行简单叙述。

8.2.1 选择语句

选择语句即条件判断控制语句，它首先判断给定的条件是否满足，然后根据判断的结果决定执行给出的若干种选择之一。C51 中选择语句有 if 语句、switch/case 语句。

1．if语句

C51 提供三种形式的 if 语句：

（1）if（表达式）{语句；}

例　if (p1!=0)

　　{c=20；}

（2）if（表达式）{语句 1；} else {语句 2；}

例如，if (p1!=0)

{c=20；}

else

{c=0；}

（3）if（表达式 1）{语句 1；}

else if（表达式 2){语句 2；}

else if（表达式 3){语句 3；}

……

else if（表达式 n){语句 n；}

else {语句 n+1；}

例如，if (a>=1) {c=10；}

else if (a>=2) {c=20；}

else if (a>=3) {c=30；}

else if (a>=4) {c=40；}

else {c=0；}

（4）if 语句的嵌套：在 if 语句中又含有一个或多个 if 语句，这种情况称为 if 语句嵌套。If 语句嵌套的基本形式如下：

外层嵌套 if 语句
- if()
  - if(){语句 1；}
  - else if(){语句 2；}

  } 内层嵌套 if 语句
- else
  - if(){语句 3；}
  - else if(){语句 4；}

  } 内层嵌套 if 语句

请注意 if 与 else 的对应关系。else 总是与它前面最近的一个 if 语句相对应，如上所示。最好使内层嵌套的 if 语句也包含 else 部分（不要省略），这样，程序中 if 的数目与 else 的数目一一对应，不致出错。另外在编程时最好使用相同深度的缩进排写的形式将同一层次上的 if-else 语句在同一列的位置上对齐，这样不仅不易出错，也便于阅读程序。

例 8.1 如图 8.1 所示，单片机 P1 口的 P1.0 和 P1.1 各接一个开关 K1、K2，P1.4、P1.5、P1.6 和 P1.7 各接一只发光二极管。有 K1 和 K2 的不同状态来确定哪个发光二极管被点亮。如表 8.5 所示。

表 8.5 发光二极管与开关状态的对应关系

K2	K1	亮的二极管
0	0	VD1
0	1	VD2
1	0	VD3
1	1	VD4

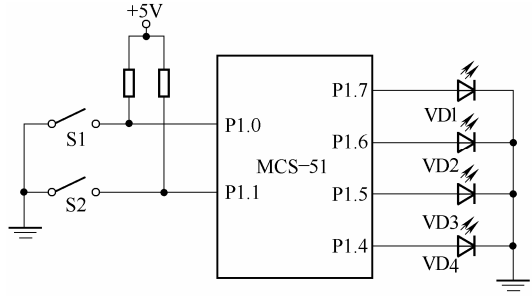


图 8.1

解：程序如下：

```
#include "reg51.h"
void main()
{
  char a;
  a=P1;
  a=a&0x03;                    //屏蔽高 6 位
  if (a==0) P1=0x83;
    else if a==1) P1=0x43;
      else if (a==2) P1=0x23;
        else P1=0x13;
}
```

2．switch/case语句

switch/case 语句是多分支选择语句，它的一般形式如下：

```
switch（表达式）
      {
      case  常量表达式 1：语句 1；  break;
      case  常量表达式 2：语句 2；  break;
      ……
      case  常量表达式 n：语句 n；  break;
      default ：语句 n+1;
}
```

（1）当 switch 括号中的表达式的值与某一 case 后面的常量表达式的值相同时，就执行它后面的语句，然后执行 break 语句，退出 switch 语句。若所有的 case 中的常量表达式的值都没有与表达式的值相匹配时，就执行 default 后面的语句。

（2）每一个 case 的常量表达式必须是互不相同的，否则将出现混乱局面。

（3）各个 case 和 default 出现的次序，不影响程序的执行结果。

（4）如果在 case 语句中遗忘了 break，则程序执行了本行之后，不会按规定退出 switch 语句，而是将执行后续的 case 语句。

例 8.2　将例 8.1 用 switch/case 语句改写。

解：程序如下：

```
#include "reg51.h"
void main()
{
  char a;
  a=P1;
  a=a&0x03;          //屏蔽高 6 位
  switch (a)
  {
      case0：P1=0x83；break；
```

```
            case1：P1=0x43；break；
            case2：P1=0x23；break；
            case3：P1=0x13；
        }
    }
```

8.2.2 循环语句

C51 循环控制的语句有 while、do-while、for 和 goto 语句，具体用法各有不同。

1．if语句和goto语句

goto 语句为无条件转向语句，它的一般形式是：

goto 语句标号；

语句标号是一个标识符，原则上任何一语句都可以有标号，标号和语句用“：”号分开。

例 8.3 例 8.2 的程序只能执行一遍，如果我们需要随时改变开关状态，进而改变二极管的发光状态，就需要不停地执行程序，现用 goto 语句构成死循环。

解：程序如下：

```
#include "reg51.h"
void main()
{
    char a；
loop：a=P1；
    a=a&0x03； //屏蔽高 6 位
    switch (a)
    {
        case0：P1=0x83；break；
        case1：P1=0x43；break；
        case2：P1=0x23；break；
        case3：P1=0x13；
    }
    goto loop；
}
```

由例 8.3 看出，goto 语句可以实现程序无条件转向。goto 语句也可以与 if 语句构成循环结构，具体实现有 2 种结构形式。

（1）loop：if（表达式）

{语句

goto loop；

}

（2）loop：{语句

if（表达式)goto loop；

}

但是多数 C 程序员都不太使用 goto 语句，原因是过多使用它会使程序结构不清晰，使程

序失去 C 语言易模块化的优点。

2．while语句

while 语句可以理解为“当条件为真时执行后面的语句”。其一般格式是：

while（条件表达式)语句；

条件表达式可以是任何表达式，语句可以是复合语句。

while 语句的执行过程：

（1）计算条件表达式的值；

（2）若其值为 1，则执行内嵌语句（循环）；若其值为 0，则退出 while 循环。

例 8.4 将例 8.3 的死循环用 while 循环实现。

解：程序如下：

```
#include "reg51.h"
void main()
{
  char a;
  while (1)
  {
      a=P1;
      a=a&0x03;        //屏蔽高 6 位
      switch (a)
  {
      case0：P1=0x83；break；
      case1：P1=0x43；break；
      case2：P1=0x23；break；
      case3：P1=0x13；
      }
  }
}
```

3．do-while语句

While 语句是先判断条件是否成立，再执行循环体；而 do-while 语句则是先执行循环体，再根据条件判断是否推出循环。其一般格式是：

do 语句 while （条件表达式)；

do-while 语句的执行过程为：

（1）执行内嵌的语句；

（2）计算表达式当表达式的值为非 0 时，则循环；当表达式的值为 0 时，则结束循环，执行 do-while 语句下面的语句。

例 8.5 将例 8.4 用 do-while 语句改写。

解：程序如下：

```
#include "reg51.h"
void main()
```

```
{
  char a;
  do {
      a=P1;
      a=a&0x03;        //屏蔽高 6 位
      switch (a)
      {
          case0：P1=0x83；break；
          case1：P1=0x43；break；
          case2：P1=0x23；break；
          case3：P1=0x13；
      }
  } while (1)；
}
```

4．for语句

for 语句的一般形式为：

for（[初值设定表达式]；[循环条件表达式]；[条件更新表达式]）语句

它的执行过程是：

（1）代入初值。

（2）判断循环条件是否为真，若其值非 0，则执行循环体并更新条件。

（3）再判断循环条件是否为真……直到条件为假时，则退出循环。

例 8.6　求 1～100 的累加和。

解：程序如下：

```
main ()
{
  float sum=0；
  int n；
  for (n=1；n<=100；n++)
      {
          sum=sum+(float)n；
      }
}
```

for 语句中的表达式是可选的，因此它的变化形式有多种。若三个表达式都没有，则相当于一个死循环。

例 8.7　将例 8.5 用 for 语句改写。

解：程序如下：

```
#include "reg51.h"
void main()
{
  char a；
```

```
    for (;  ; )
        {
            a=P1;
            a=a&0x03;    //屏蔽高 6 位
            switch (a)
            {
                case0：P1=0x83；break；
                case1：P1=0x43；break；
                case2：P1=0x23；break；
                case3：P1=0x13；
            }
        }
}
```

8.3 C51 构造数据类型

8.3.1 数组

数组是相关数据的一个有序集合，数组中的每个元素都是同一类型的数据。数组集合用一个名字来标识，称为数组名。数组中元素的顺序用下标表示，下标表示该元素在数组中的位置。下标为 n 的元素可以表示为数组名[n]。改变[ ]中的下标就可以访问数组中所有的元素。一个数组元素等同于一个变量，因此又可以说数组是一组相同数据类型的相关变量的有序集合。

1．一维数组

由具有一个下标的数组元素组成的数组称为一维数组。

（1）一维数组的定义：一维数组定义的一般形式是：

类型说明符　数组名[元素个数]；

数组名是一个标识符，元素个数是一个常量表达式，不能是含有变量的表达式。例如，

```
int a[50];                          //定义一个数组名为 a 的数组，数组包含 50 个整型的元素
```

（2）一维数组的初始化：在定义数组时可以对数组整体初始化，若定义后想对数组赋值，则只能对每个元素分别赋值。例如。

```
int a[5]={1，2，3，4，5}；          //给全部元素赋值，a[0]=1，a[1]=2，a[2]=3，a[3]=4，a[4]=5
int b[6]={1，2，6}；                //给部分元素赋值，b[0]=1，b[1]=2，b[2]=6，b[3]=b[4]=b[5]=0
int d[10]；d[0]=4；d[1]=-6；……     //定义完后再赋值
```

2．二维数组

由具有两个下标的数组元素组成的数组称为二维数组。

（1）二维数组的定义：二维数组定义的一般形式是：

类型说明符　数组名[行数] [列数]；

数组名是一个标识符，行数和列数都是常量表达式。例如，

```
float a[3][4]；                     //a 数组有 3 行 4 列共 12 个实型元素
```

（2）二维数组的初始化：与一维数组的初始化相似，定义时可以整体初始化，也可以在定义后单个地进行赋值。例如，

```
int a[3][4]={{1，2，3，4}，{5，6，7，8}，{9，10，11，12}}；//全部初始化
int a[3][4]={{1，2，3，4}，{5，6，7，8}，{}}；//部分初始化，a[2][0]= a[2][1]= a[2][2]= a[2][3]=0
```

C51 也可以定义多维数组。

3．字符数组

若一个数组的元素是字符型的，则该数组就是一个字符数组。

（1）字符数组的定义与赋值：与一维数组的定义赋值的方法类似，例如，

```
char a[12]={ "Chong Qing"}；
```

（2）字符串和字符串结束符：C 语言中没有字符串变量，需用字符数组来处理字符串。当数组中存放的实际字符个数与数组的长度不一样时，为了测定字符串的实际长度和使用系统提供的各种字符串函数，C 语言规定了字符串结束标志'\0'，它是一个 ASCII 码值为 0 的字符。在一个字符数组中，一旦遇到字符'\0'，就表示字符串结束，其后的字符忽略不计。上面 a 数组的后 2 个元素皆为'\0'，字符串常量中也自动包含一个字符串结束符'\0'。

4．查表

数组的一个很有用的用途就是查表。在单片机应用中，常常要对数学公式进行计算以及对一些传感器的非线性特性进行补偿，这时，采用查表的办法就比较简单有效。因为单片机的计算能力有限，可以将复杂的数学公式或补偿算法事先计算成表格，存入程序存储器中，而这个表格就是数组。

8.3.2 指针

关于指针的基本概念请参阅有关资料，下面就 C51 的指针类型作些说明。

C51 支持“基于存储器”的指针和“一般”指针。当定义一个指针变量时，若未指定它所指向的对象的存储类型，则该指针变量被认为是一般指针；反之若指定了它所指对象的存储类型，则该指针被认为是基于存储器的指针。

基于存储器的指针类型由 C 源代码中指定的存储类型决定，并在编译时确定，这种指针只需 1～2 个字节，并且高效。

一般指针需占 3 个字节：第一个字节为存储器类型的编码（由编译模式的默认值确定)，剩余两个字节为地址偏移量。存储器类型决定了对象所用的 MCS-51 单片机存储空间，偏移量指向实际地址。一个“一般”指针可以访问任何变量而不管它存储空间的哪个位置。这就允许一般函数，如 memcpy()等将数据从任意一个地址拷贝到另一个地址空间。

（1）基于存储器的指针：基于存储器的指针是在说明一个指针时，指定它所指向的对象的存储类型。例如，

```
char xdata *px；
```

px 为指向一个定义在 xdata 存储器中的字符变量的指针变量。px 本身在默认的存储器区域（由编译模式决定），其长度为 2 字节。

前面已经讲到，C51 有三种存储模式：即 SMALL、COMPACT 和 LARGE。C51 的存储模式确定函数的参数与局部变量的存储区域。在 SMALL 模式下，函数的参数与局部变量位

于单片机的内部 RAM，在 COMPACT 和 LARGE 模式下，函数的参数与局部变量位于单片机外部 RAM。例如，

char xdata *data py；

py 为指向一个定义在 xdata 存储器中的字符变量的指针变量。py 本身在 RAM 中，与编译模式无关，其长度也为 2 字节。

（2）一般指针：在函数的调用中，函数的指针参数需要用一般指针。一般指针的说明形式如下：

char *pz；

这里没有指定指针变量 pz 所指向的变量的存储类型，pz 处于编译模式默认的存储区，长度为 3 字节，格式为：

地址	+0	+1	+2
内容	存储类型的编码	高位地址偏移量	低位地址偏移量

其中存储类型由编译模式决定，不同的存储区域的编码如下：

存储类型	idata	xdata	pdata	data	code
编码值	1	2	3	4	5

在用常量指针时，如定义外部端口的地址，必须注意正确定义存储类型和偏移量。

例如，要将数值 0x41 写入地址为 0x8000 的外部数据存储器中。则可如下写为：

```
#include "absacc.h"
XBYTE[0x8000]=0x41；
```

其中 XBYTE 是一个指针，它是在头文件 absacc.h 中定义的，定义如下：

```
#define XBYTE ((unsigned char *) 0x2000L)
```

XBYTE 被定义为(unsigned char *) 0x2000L，为一般指针，其存储类型为 2，即为 xdata 型，偏移量是 0000，这样，XBYTE 成为指向外部数据存储器的零地址单元的指针。而 XBYTE[8000]则表示外部数据存储器的 0x8000 单元。

C51 与标准 C 类似，它的构造数据类型还有结构体、共用体和枚举等。请读者参阅有关资料，在此就不再叙述了。

8.4 C51 函数

8.4.1 函数的定义与分类

1．函数的分类

从 C 语言程序的结构上划分，C 语言函数分为主函数 main()和普通函数两种。

而对于普通函数，从不同的角度或以不同的形式又可以分为：标准库函数和用户自定义函数。

（1）标准库函数：标准库函数是由 C 编译系统的库函数提供的，在 C 编译系统中将一些独立的功能模块编写成公用函数，并将它们集中存放在系统的函数库中，供程序设计时使用。故把这种函数称为标准库函数。C51 也提供了比较丰富的库函数资源，将在后面对常用的库函数的功能进行说明。

（2）用户自定义函数：用户自定义函数是用户根据自己的需要而编写的函数。

从函数定义的形式上可划分为：无参数函数、有参数函数和空函数。

无参数函数：此种函数被调用时，既无参数输入，也不返回结果给调用函数，它是为完成某种操作而编写的。

有参数函数：在调用此种函数时，必须提供实际的输入参数，必须说明与实际参数一一对应的形式参数，并在函数结束时返回结果供调用它的函数使用。

空函数：此种函数体内无语句，是空白的。调用此中函数时，什么工作也不做。而定义此种函数的目的并不是为了执行某种操作，而是为了以后程序功能的扩充。

2．函数的定义

函数定义的一般形式为：

```
返回值类型　函数名（形式参数列表）
{
  函数体
}
```

其中，

（1）关于返回值类型：

① 可以是基本数据类型（int，char，float，double 等）及指针类型。

② 当函数没有返回值时，用标识符 void 说明该函数没有返回值。

③ 若没有指定返回值类型，默认返回值为整型类型。

④ 一个函数只能有一个返回值，该返回值是通过函数中的 return 语句获得的。

（2）函数名必须是一个合法标识符。

（3）形式参数列表包括了函数所需全部参数的定义。此时函数的参数称为形式参数，简称形参。形参可以是基本数据类型的数据、指针类型数据、数组等。在没有调用函数时，函数的形参和函数内部的变量未被分配内存单元，即它们是不存在的。

（4）函数体由两部分组成：函数内部变量定义和函数体其他语句。

（5）各函数的定义是独立的。

（6）函数的定义不能在另一个函数的内部。

8.4.2　函数的调用

函数调用的一般形式为：

```
函数名（实际参数列表）;
```

在一个函数中需要用到某个函数的功能时，就调用该函数。调用者称为主调函数，被调用者称为被调函数。若被调函数是有参函数，则主调函数必须把被调函数所需的参数传递给被调函数。传递给被调函数的数据称为实际参数，简称实参。若被调函数是无参函数，则调用该函数时，可以没有参数列表，但括号不能省。被调函数执行完后再返回主调函数继续执行剩余程序。实参与形参在数量、类型和顺序上都必须一致；实参可以是常量、变量和表达式；实参对形参的数据传递是单向的，即只能将实参传递给形参。

例 8.8　编写程序求三个整数中的最大值。

解：程序如下：

```
int max(int x，int y，int z)
  { int a=x；
      if(y>x) a=y；
      if(z>a) a=z；
      return a；
  }

main()
{ int x1，y1，z1；
  printf("Enter 3 numbers\n")；
  scanf("%d %d %d"，&x1，&y1，&z1)；
  printf("The max is %d\n"，max(x1，y1，z1))；
}
```

例中，（1）函数 max 有三个形参 x、y、z，都是整型参数；函数 main 有三个实参 x1、y1、z1，也都是整型。

（2）变量 a 是函数 max 的内部变量，只在函数 max 内有效。

（3）函数的返回值是通过语句 return a；实现的。

（4）一个函数只能返回一个值，可以有多个 return 语句，但只有一个被执行。

（5）函数的形参和内部变量可以与其他函数的形参和内部变量同名。

C51 的库函数中包含了 printf、scanf 函数，该函数格式化字符串，把它们输入或输出到标准输入或输出设备，对 PC 机来说标准输入、输出设备就是键盘和显示设备，对 8051 单片机来说就是串行口。

8.4.3 函数的嵌套调用与递归调用

函数的嵌套调用就是在调用一个函数的过程中，又调用另一个函数。

函数的递归调用就是一个函数在其函数体内调用自己。递归调用是一种特殊的循环结构。在 C51 编程中，递归函数必须是可重入的，可重入的函数必须加关键字 reentrant。

例 8.9 求 n!

解：设函数 f1(n)=n！，则 f1(n)可以用如下的递归公式来表示：

$$f1(n)=\begin{cases} n\times f1(n-1) & (n>1) \\ 1 & (n=0,\ 1) \end{cases}$$

由递归公式，可以很快得到下面的递归程序：

```
float f1(int n)reentrant
  { if(n==0||n==1)return 1；
  return   n* f1(n−1)；
  }
main()
{
  printf("%d\n"，f1(4))；
}
```

上述程序为求 4！的程序。

8.4.4 指向函数的指针变量

在把程序调入内存运行时，每一个函数都被分配了内存单元。我们将函数的第一条指令所在的地址单元称为该函数的入口地址，可以定义一个指针变量来存放函数的地址，然后通过该指针变量就可调用此函数。

指向函数的指针变量的定义的一般形式：

类型说明符 （*指针变量名） （参数列表）；

其中，类型说明符指定了指针所指函数的返回值类型，形参列表指定了指针所指函数的参数个数及类型。

例如，有如下的函数定义：

```
double max(double x，double y)
{
  函数体
}
```

可以定义指针变量 p 指向该函数。定义如下：

```
double(*p)(double，double)；
```

一旦定义了一个指向某类函数的指针变量后，这个指针变量就只能指向该类函数，即返回值相同、参数的个数、类型、顺序都相同的一类函数，而不能是任意的函数。

例 8.10 实现一个简单的计算器，能完成加、减、乘、除和取余运算。

解：程序如下：

```
int add(int x，int y)                          //add 函数实现加法功能
{ return x+y；}
int sub(int x，int y)                          //sub 函数实现减法功能
{ return x−y；}

int mul(int x，int y)                          //mul 函数实现乘法功能
{ return x*y；}
int div(int x，int y)                          //div 函数实现除法功能
{ return x/y；}

int mod(int x，int y)                          //mod 函数实现取余功能
{ return x%y；}

main()
{ int x，y，c；
   char z；
   int(*p)(int，int)；                         //p 定义为指向函数的指针
while(1)
{ printf("Input the first number：\n")；
 scanf("%d"，&x)；                             //输入第一个数
   printf("Input the second number：\n")；
```

```
    scanf("%d"，&y)；                          //输入第二个数
    printf("Choice the operator：1 +；2 −；3 *；4/；5 %\n")；
    scanf("%d"，&z)；                          //输入运行代码
  switxch(z)
    {
      case 1：p=add；c='+'；break；            //将 add 函数地址赋给 p
      case 2：p=sub；c='−'；break；            //将 sub 函数地址赋给 p
      case 3：p=mul；c='*'；break；            //将 mul 函数地址赋给 p
      case 4：p=div；c=' / '；break；          //将 div 函数地址赋给 p
      case 5：p=mod；c='%'；break；            //将 mod 函数地址赋给 p
    }
  printf("%d %c %d=%d\n"，x，c,(*p)(x，y))；  //调用 p 所指函数
  printf("Continue? 1 YES；  0 NO\n")；
  scanf("%d"，&x)；
  if(!x) break；
    }
  }
```

8.4.5 C51 的库函数

C51 编译器提供了丰富的库函数，使用这些库函数大大提高了编程效率，用户可以根据需要随时调用。每个库函数都在相应的头文件中给出了函数的原型，使用时只需在源程序的开头用编译预处理命令#include 将相关的头文件包含进来即可。下面就一些常用的 C51 库函数做一些介绍。

1．字符函数库ctype.h

（1）extern bit isalpha(char)；

检查参数字符是否为英文字母，是则返回 1，否则返回 0。

（2）extern bit isalnum(char)；

检查参数字符是否为英文字母或数字字符，是则返回 1，否则返回 0。

（3）extern bit iscntrl(char)；

检查参数字符是否为控制字符，即 ASCII 码值为 0x00～0x1f 或 0x7f 的字符，是则返回 1，否则返回 0。

（4）extern bit islower(char)；

检查参数字符是否为小写英文字母，是则返回 1，否则返回 0。

（5）extern bit isupper(char)；

检查参数字符是否为大写英文字母，是则返回 1，否则返回 0。

（6）extern bit isdigit(char)；

检查参数字符是否为数字字符，是则返回 1，否则返回 0。

（7）extern bit isxdigit(char)；

检查参数字符是否为 16 进制数字字符，是则返回 1，否则返回 0。

（8）extern char toint(char)；

将 ASCII 字符的 0～9、a～f（大小写无关）转换为 16 进制数字。

（9）extern char toupper(char)；

将小写字母转换成大写字母，如果字符不在“a～z”之间，则不作转换直接返回该字符。

（10）extern char tolower(char)；

将大写字母转换成小写字母，如果字符不在“A～Z”之间，则不作转换直接返回该字符。

2．标准函数库stdlib.h

（1）extern float atof(char*s)；

将字符串 s 转换成浮点数值并返回它。参数字符串必须包含与浮点数规定相符的数。

（2）extern long atol(char*s)；

将字符串 s 转换成长整型数值并返回它。参数字符串必须包含与长整型数规定相符的数。

（3）extern int atoi(char*s)；

将字符串 s 转换成整型数值并返回它。参数字符串必须包含与整型数规定相符的数。

（4）void *malloc(unsigned int size)；

返回一块大小为 size 个字节的连续内存空间的指针。如果返回值为 NULL，则无足够的内存空间可用。

（5）void free(void *p)；

释放由 malloc 函数分配的存储器空间。

（6）void int_mempool(void *p，unsigned int size)；

清零由 malloc 函数分配的存储器空间。

3．数学函数库math.h

（1）extern int abs(int val)；

extern char abs(char val)；

extern float abs(float val)；

extern long abs(long val)；

计算并返回 val 的绝对值。这四个函数的区别在于参数和返回值的类型不同。

（2）extern float exp(float x)；

返回以 e 为底的 x 的幂，即 e^x。

（3）extern float log(float x)；

extern float log10(float x)；

log 返回 x 的自然对数，即 lnx；log10 返回以 10 为底的 x 的对数，即 $\log_{10}x$。

（4）extern float sprt(float x)；

返回 x 的正平方根。

（5）extern float sin(float x)；

extern float cos(float x)；

extern float tan(float x)；

sin 返回值为 sin(x)；cos 返回值为 cos(x)；tan 返回值为 tan(x)。

（6）extern float pow(float x，float y)；

返回值为 x^y。

4．绝对地址访问头文件absacc.h

（1）#include CBYTE((unsigned char*)0x50000L)
#include DBYTE((unsigned char*)0x40000L)
#include PBYTE((unsigned char*)0x30000L)
#include XBYTE((unsigned char*)0x20000L)
用来对 MCS-51 系列单片机的存储器空间进行绝对地址访问，以字节为单位寻址。
CBYTE 寻址 CODE 区；
DBYTE 寻址 DATA 区；
PBYTE 寻址 XDATA 的 00H～FFH 区域（用 MOVX @Ri 指令）；
XBYTE 寻址 XDATA 区（用 MOVX @DPTR 指令）。
（2）#include CWORD((unsigned int*)0x50000L)
#include DWORD((unsigned int*)0x40000L)
#include PWORD((unsigned int*)0x30000L)
#include XWORD((unsigned int*)0x20000L)
与前面的宏定义相同，只是数据为双字节。

5．内部函数库intrins.h

（1）unsigned char_crol_(unsigned char val，unsigned char n)；
unsigned int_irol_(unsigned int val，unsigned char n)；
unsigned long_lrol_(unsigned long val，unsigned char n)；
将变量 val 循环左移 n 位。
（2）unsigned char_cror_(unsigned char val，unsigned char n)；
unsigned int_iror_(unsigned int val，unsigned char n)；
unsigned long_lror_(unsigned long val，unsigned char n)；
将变量 val 循环右移 n 位。
（3）void_nop_(void)；
该函数产生一个单片机的 NOP 指令，用于延时一个机器周期。
（4）bit_testbit_(bit x)；
测试给定的位参数 x 是否为 1，为 1，则返回 1，同时将该位复位为 0；否则返回 0。

6．访问SFR和SFR_bit地址头文件reg××.h

头文件 reg51.h 和 reg52.h 中定义了 MCS-51 系列单片机的 SFR 寄存器名和相关的位变量名。

8.5 C51 应用编程实例

8.5.1 MCS-51 系列单片机内部资源编程

1．外部中断服务程序及例程

C51 为中断服务程序的编写提供了方便。C51 的中断服务程序是一种特殊的函数，它的说明形式为：

void　函数名(void) interrupt n [using m]
{函数体语句}

这里 interrupt 和 using 是为编写 C51 中断服务程序而引入的关键字，interrupt 是不可缺少的，它表示该函数是一个中断服务函数，interrupt 后的整数 n 表示该中断函数是对应中断源的编号，n 的取值范围为 0～31，但具体的中断号要取决于芯片的型号，如 AT89C52 实际上只使用了 0～5 号中断，如表 8.6 所示。

表 8.6　AT89C52 的中断编号

中　断　源	中 断 编 号
外部中断 0	0
定时器 T0	1
外部中断 1	2
定时器 T1	3
串行口中断	4
定时器 T2	5

using 指定该中断服务程序要使用的工作寄存器组号，m 为 0～3。若不使用关键字 using，则编译系统会将当前工作寄存器组的 8 个寄存器都压入堆栈。

关键字 interrupt 和 using 只能用于中断服务函数的说明，而不能用于其他函数。

程序中的任何函数都不能调用中断服务函数，中断服务函数是由系统调用的。

例 8.11　外部中断 0 引脚(P3.2)接一个开关，P1.0 接一个发光二极管。开关闭合一次，发光二极管改变一次状态。

解：程序如下：

```
#include "reg51.h"
#include "intrins.h"
void delay(void)
  {
      int a=5000;
      while(a--)_nop_();
  }
void int_srv(void)interrupt 0 using 1
  {
      Delay();                    //调用延时子函数
      if(INT0==0)
          {P10=!P10;              //如果 P3.2=0，P10 取反
          while(INT0==0);  }      //如果 P3.2=0，等待，直到 P3.2=1，中断返回
  }
void main()
  { P10=0;
      EA=1;                       //开中断
      EX0=1;
      While(1);                   //等待
  }
```

2．定时器/计数器编程

例 8.12　从 P1.0 输出方波信号，周期为 50ms。设单片机的 f_{osc}=6MHz。

解：利用单片机定时器产生方波信号，定时时间为 25ms。当单片机的 f_{osc}=6MHz，用 T0 工作于方式 1，最大定时时间约为 32ms，能够满足要求。为了计算方便，定时器的计数初值为：

$a=2^{16}-0.025\times6000000/12=53036$

得到：TH0=0xCFH，TL0=0x2CH

程序如下：

```
#include "reg51.h"
void main()
{
    TMOD=0x01;                  //定时器 T0，方式 1
    TH0= 0xCFH;                 //赋计数初值
    TL0=0x2CH;
    ET0=1;                      //开中断
    EA=1;
    TR0=1;                      //启动定时器
    While(1);                   //等待
}
void T0_srv(void)interrupt 1 using 1
{
    TH0= 0xCFH;
    TL0=0x2CH;
    P10=!P10;
}
```

8.5.2 MCS-51 系列单片机扩展资源编程

例 8.13 如图 8.2 所示，单片机通过 74LS164 实现串口转并口，控制 8 只发光二极管以流水方式亮灭，并一直循环往复，设 f_{osc}=11.0592MHz。

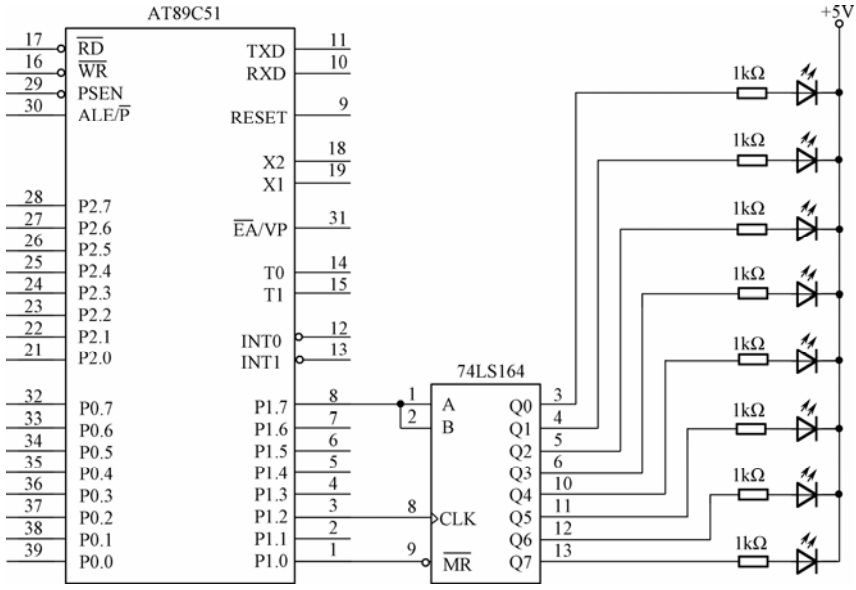


图 8.2　利用 74LS164 实现串并口转换

74LS164 是 8 位串入并出的移位寄存器，真值表如表 8.7 所示。

表 8.7　74LS164 真值表

输　入			输　出
$\overline{MR}$	CLK	A　B	Q_0 Q_1 …Q_7
L	*	*　*	L　L … L
H	L	*　*	保持
H	↑	H　H	H Q_{0n}…　Q_{6n}
H	↑	L　*	L Q_{0n}…　Q_{6n}
H	↑	*　L	L Q_{0n}…　Q_{6n}

注：表中，*——任意状态；H——高电平；L——低电平；Q_{0n}…　Q_{6n}——时钟上升沿时输出 Q_0…　Q_6 前一次状态。

解：图 8.2 中所示由 P1.7 送出数据，移位时钟由 P1.2 送出，在移位时钟作用下，P1.7 口发送的数据一位一位地移入 74LS164 中，程序如下：

```
#include <reg52.h>
#include <intrins.h>
#define uchar unsigned char
#define uint unsigned int
sbit data_164 = P1^7;
sbit clk_164 = P1^2;
sbit mr_164 = P1^0;
void delayms(uint t)                    //延时 t 毫秒
{
  uchar k;
  while(t--)
  {
      for(k=0;  k<125;  k++) {;}
  }
}
void wr_bits(uchar num)                 // 写数据子函数
{ uchar i;
  for(i=0; i<8; i++)
  { if(num&0x80) data_164 =1;
      else   data_164 =0;
      num<<=1;                          //向左移动一位，先送高位
      clk_164 =0;                       //下降沿将数据写入 164
      _nop_();
      clk_164 =1;
  }
}
void main()
```

```
{ uchar n,temp;
  P1 = 0xff;
  clk_164 =0;
  mr_164 =0;
  delayms(1);
  mr_164 =1;
  while(1)
  { temp=0xfe;                          //赋显示初值
      for(n=0; n<8; n++)
          { wr_bits(temp);              //写数据，送显示
            delayms(400);
            temp<<=1;                   //准备下一个显示数据
            temp=temp|0x01;             //指令屏蔽,保留最低位
      }
      wr_bits(0xff);                    //关闭显示
      delayms(400);
  }
}
```

8.5.3 MCS-51 系列单片机接口技术编程

例 8.14 如图 8.3 所示独立键盘，现用 C51 来实现 4 个按键分别控制 4 个 LED 的亮灭，设 f_{osc}=11.0592MHz。

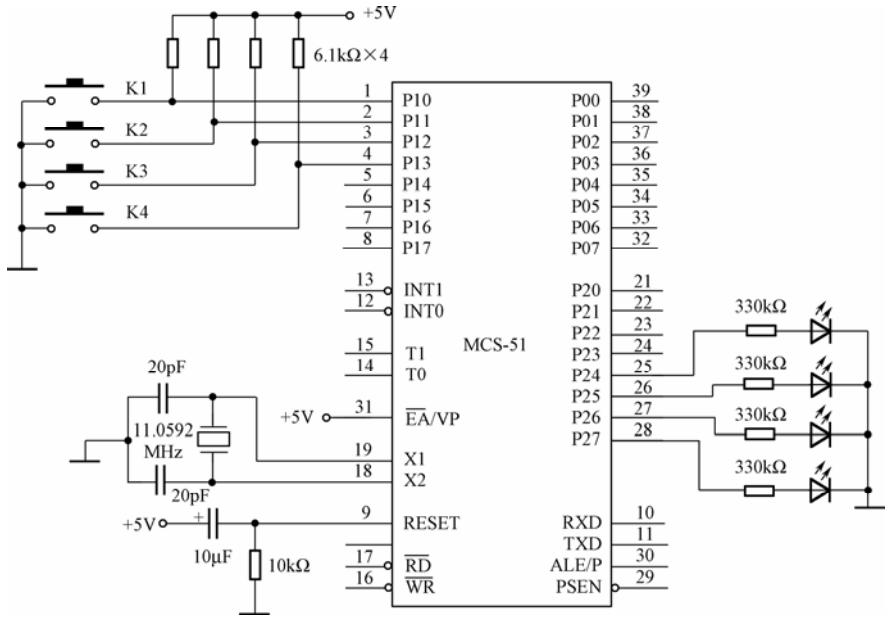


图 8.3 独立按键接口电路

解：根据题意要求，假设 P1.0 对应的按键去控制接在 P2.7 上的 LED，按依次按键，所对应 LED 的显示状态发生一次改变。依次是 P1.1 对应 P2.6，P1.2 对应 P2.5， P1.3 对应 P2.4，

定时中断去扫描按键并进行 LED 控制，则程序如下：

```
#include "reg52.h"
#define THCO     0xee                          //定时时间常数
#define TLCO     0x0
sbit P10=P1^0;
sbit P11=P1^1;
sbit P12=P1^2;
sbit P13=P1^3;
sbit P27=P2^7;
sbit P26=P2^6;
sbit P25=P2^5;
sbit P24=P2^4;
void main()
{
      TMOD=0x01;
      TH0=THCO;
      TL0=TLCO;
      TR0=1;
      ET0=1;
      EA=1;
      IT0=1;
      P1=0x0f;
      while(1) ;
}
void timer0() interrupt 1                       //定时器 T0 中断服务程序
{
 static unsigned char count=0;
 TH0=THCO;
 TL0=TLCO;
  if(P10==0||P11==0||P12==0||P13==0)            //只要有任意键被按下，则处理
 {
      count++;                                  //消除按键抖动
      if(count>=30)
      {
          count=0;
          if(P10==0)     P27=!P27;              //改变 LED 显示状态
          else   if(P11==0)   P26=!P26;
          else   if(P12==0)   P25=!P25;
          else   if(P13==0)   P24=!P24;
      }
  }
```

```
    else count=0;
}
```

例 8.15 如图 6.13 所示的动态显示电路，现用 C51 编程，在 4 个数码管上分别显示 1、2、3、4，设 f_{osc}=11.0592MHz。

解：程序如下：

```
#include "reg52.h"
#define THCO    0xee
#define TLCO    0x0
unsigned char   code Duan[]={0x3F, 0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F}; //字段码
unsigned char   Data_Buffer[4]={1,2,3,4};
sbit P10=P1^0;
sbit P11=P1^1;
sbit P12=P1^2;
sbit P13=P1^3;
void main()
{
    TMOD=0x01;
    TH0=THCO;
    TL0=TLCO;
    TR0=1;
    ET0=1;
    EA=1;
while(1) ;
}
void timer0() interrupt 1                    //5ms 定时中断服务子程序
{
  static unsigned char Bit=0;
  TH0=THCO;
  TL0=TLCO;
  Bit++;
  if(Bit>=4)Bit=0;
  P1|=0x0f;                                  //数码管全灭
  P2=Duan[Data_Buffer[Bit]];                 //送段码
  switch(Bit)                                //送位码
  {
  case 0: P10=0; break;
  case 1: P11=0; break;
  case 2: P12=0; break;
  case 3: P13=0; break;
  }
```

例 8.16 如图 6.18 所示，对 ADC0809 的 8 个通道的模拟量轮流采集一次，采集的结果

放在数组 ad 中。

解：程序如下：

```
#include "reg51.h"
#include "absacc.h"
sbit ad_busy=P3^2;
main()
{       static char idata ad[8];
        char i;
        char pdata *ad_ch;
        ad_chl=0x78;                        //设置初始通道地址
        for(i=0; i<8; i++)
        {   *ad_chl=0;              //启动 A/D 转换
            i=i;                    //等待 EOC 信号变低
            i=i;
            while(ad_busy==1);  //查询
            ad[i]=*ad_chl;          //存放结果
            ad_chl++ ;
        }
}
```

例 8.17　如图 6.23 所示，将 AD574A 的转换结果放入单片机片内单元中。

解：由图可知，AD574 命令口地址为 0x00，低字节的口地址为 0x03，高字节的口地址为 0x01。程序如下：

```
#include "reg51.h"
#include "absacc.h"
#define ADCOM    PBYTE[0x00]
#define ADLO    PBYTE[0x03]
#define ADHI    PBYTE[0x01]
sbit ad_busy=P3^2;
main()
{       int idata result;
        ADCOM=0;                                        //启动 A/D 转换
        While(ad_buy==1);                               //查询
        result=(int)((ADHI)*256+((ADLO>>4)&0x0f));  //存结果
}
```

例 8.18　图 8.4 为单片机与串行 A/D 转换芯片 TCL594 的硬件连接图，编写程序，通过单片机的 P1 口控制 TCL594 实现模拟电压信号的采集。

解：程序如下：

```
#include   "reg51.h"
#include   "absacc.h"
#include   "intrins.h"
sbit   CS=P1^4;
```

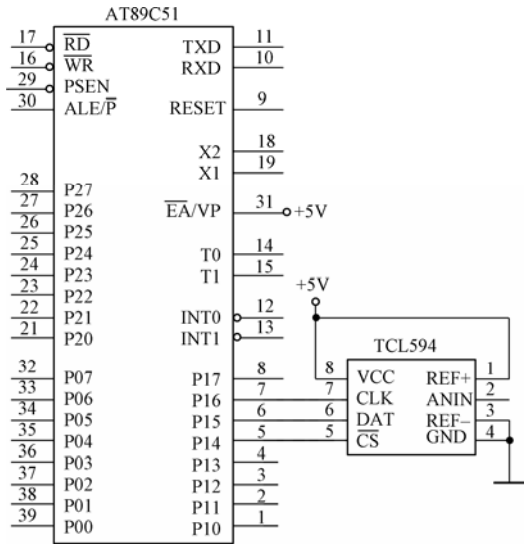


图 8.4　单片机与 TCL594 的接口电路

```
sbit    DAT=P1^5;
sbit    CLK=P1^6;
unsigned char save=0;
unsigned char ADD;
void    ad()                          //A/D 转换子程序
{    bit B;
        char    count=8;
        save=0;
        CLK=0;
        CS=0;
for(; count>0; count--)
        {    CLK=1;
        _nop_();    _nop_();
        B=DAT;
        save<<=1;
        if(B==1)
        save=save|0x01;
        CLK=0;
        _nop_();
        }
ADD=save;
CS=1;
CLK=1;
}
main()
{
```

```
while(1)
    { ad()；}                    //启动 A/D 转换
}
```

例 8.19　如图 6.28 所示，要求 DAC0832 输出锯齿波电压信号，信号周期不限，则 C51 程序如下：

```
#include "reg51.h"
#include "absacc.h"
#define DA0832   PBYTE[0x7f]
main()
{       char i;
        do {
            for(i=0；i<255；i++)
            {DAC0832=i；}
           } while(1)；
}
```

例 8.20　图 8.5 为单片机与 AT24C01 的接口电路，编写程序实现从 AT24C01 读写数据的功能。

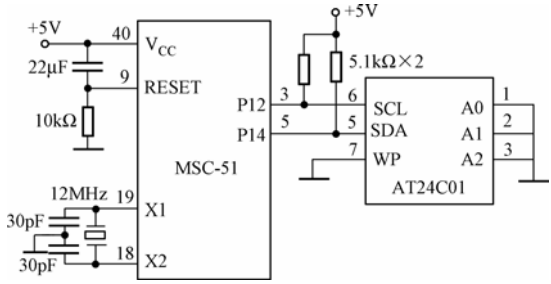


图 8.5　单片机与 AT24C01 的接口电路

解：在第 6 章已经讲过，AT24C01 是基于 I^2C 总线的 EEPROM，现在用 C51 来实现 AT24C01 的读写操作。程序如下：

```
#include <reg51.h>
#include <intrins.h>
sbit   SDA= P1^4；                    //I²C 数据传送位
sbit   SCL= P1^2；                    //I²C 时钟传送位
unsigned char i2c_read(unsigned char)；
void i2c_write(unsigned char,unsigned char)；
void i2c_send8bit(unsigned char)；
unsigned char i2c_receive8bit(void)；
void i2c_start(void)；
void i2c_stop(void)；
bit i2c_ack(void)；
void main(void)
{
```

```
        unsigned char dd;
        i2c_write(0x00,0x55);
        _nop_();
        dd=i2c_read(0x00);
 for(;;)
   {}
}
 //============== void i2c_write(地址，数据)==================
//该函数的功能是将一个字节的数据写到指定的 EEPROM 单元中。
//=====================================================
void i2c_write(unsigned char Address,unsigned char Data)
{
do{
        i2c_start();
        i2c_send8bit(0xA0);
        }while(i2c_ack());                    // i2c_ack()=1，表示无确认，再次发送
i2c_send8bit(Address);
i2c_ack();
i2c_send8bit(Data);
i2c_ack();
i2c_stop();
return;
}
//============== unsigned char i2c_ read (地址，数据)===========
//该函数的功能是将 EEPROM 指定单元中的一个字节的数据读出来。
//=====================================================
unsigned char i2c_read(unsigned char Address)
{
        unsigned char c;
        do{
             i2c_start();
             i2c_send8bit(0xA0);
        }while(i2c_ack());
        i2c_send8bit(Address);
        i2c_ack();
             do{
                  i2c_start();
                  i2c_send8bit(0xA1);
                  }while(i2c_ack());
        c=i2c_receive8bit();
        i2c_ack();
```

```
    i2c_stop();
    return(c);
}
//============== void i2c_start(void)=========================
//该函数的功能是启动 I²C 总线
//============================================================
void i2c_start(void)
{
  SDA=1;                                    //发送起始条件的数据信号
  _Nop();
  SCL=1;
  _Nop(); _Nop(); _Nop(); _Nop(); _Nop();   //起始条件建立时间大于 4.7μs，延时
  SDA=0;                                    //发送起始信号
  _Nop(); _Nop(); _Nop(); _Nop(); _Nop();   //起始条件锁定时间大于 4μs
  SCL=0;                                    //钳住 I²C 总线，准备发送或接收数据
  _Nop(); _Nop();
  return;
}
//============== void i2c_ stop (void)=========================
//该函数的功能是结束 I²C 总线
//============================================================
void i2c_stop(void)
{
    SDA=0;                                  //发送结束条件的数据信号
    _Nop();                                 //发送结束条件的时钟信号
    SCL=1;                                  //结束条件建立时间大于 4μs
    _Nop();  _Nop();  _Nop();  _Nop();  _Nop();
    SDA=1;                                  //发送 I²C 总线结束信号
    _Nop();  _Nop();  _Nop();  _Nop();
    return;
}
//================ bit i2c_ack(void)==========================
//该函数的功能是发送接收确认信号
//============================================================
bit i2c_ack(void)
{
    bit   ack;
    _Nop(); _Nop();
    SDA = 1;
    _Nop(); _Nop();
    SCL = 1;
```

```
    _Nop(); _Nop(); _Nop();
    if    (SDA==1)    ack = 1;
    else   ack = 0;
    SCL = 0;
    return  (ack);
}
//============= void i2c_send8bit(unsigned char b)==================
//该函数的功能是发送 8 位数据
//=========================================================
void i2c_send8bit(unsigned char b)
{
    unsigned  char a;
    for(a=0; a<8; a++)                                  //8 位数据
    {
        if ((b<<a)&0x80) SDA = 1;                       //判断并传送数据位
        else  SDA = 0;
        _Nop();
        SCL = 1;                                        //准备开始接收数据位
        _Nop(); _Nop(); _Nop(); _Nop(); _Nop();
        SCL = 0;
    }
    return;
}
//============ unsigned char i2c_receive8bit(void)============
//该函数的功能是接收八位数据
//=========================================================
unsigned char i2c_receive8bit(void)
    {
     unsigned  char a;
    unsigned char b=0;
    SDA=1;                                              //置数据线为输入方式
        for(a=0; a<8; a++)
          {
            _Nop();
            SCL=0;                                      //置时钟线为低准备接收数据
            _Nop();  _Nop();  _Nop();  _Nop(); _Nop();
            SCL =1;                                     //使数据线上数据有效
            _Nop();  _Nop();
            b=b<<1;
            if (SDA==1)   b=b|0x01;                     //读数据位，接收数据位放入 b 中
               SCL = 0;
```

```
            }
        return  (b);
    }
```

本 章 小 结

C51 是 MCS-51 单片机高效的开发工具，它与标准 C 语言有很多相似之处，由于 MCS-51 单片机在组成及结构上有许多自己的特点，因此 C51 也有许多不同之处。本章要求：掌握 C51 的数据类型、C51 的数据存储类型、C51 的存储模式、C51 对 SFR 的定义、C51 对位变量的定义，在这些方面与标准 C 语言有很大不同，至于其他规则与标准 C 语言是基本一致的；了解 C51 的库函数；学习时应与汇编语言与标准 C 语言的程序对照起来，特别强调 C51 如何操作基本 I/O 口、扩展 I/O 口，如何进行位操作和对特殊功能寄存器的操作，掌握 C51 对定时器/计数器的应用及中断的使用方法。

习 题 8

8.1　MCS-51 单片机的存储空间有哪些？分别有些什么存储类型？这些存储类型的值域范围是多少？

8.2　按给定的存储类型和数据类型，写出下列变量的说明形式：

up，down　整数，使用内部 RAM 单元存储；

first，last　浮点小数，使用外部 RAM 单元存储；

ch，cl　字符，使用内部 RAM 单元存储。

8.3　设系统时钟为 6MHz，利用 T0 方式 1 在 P1.6 上产生频率为 100Hz 的方波，要求定时器溢出时采用中断方式处理。

8.4　用单片机制作一个模拟航标灯，灯接在 P1.7 上，INT0 接光敏元件。使航标灯夜间间歇发光，亮 2 秒，灭 2 秒，周而复始。

8.5　利用如图 6.23 所示电路，编写 AD574A 连续采集 20 个数据，除去最大值和最小值后求其平均值的程序。

8.6　设计一个 MCS51 单片机控制的 2×8 键盘和 8 位 LED 显示电路，用 C51 编写相应的键盘扫描程序和动态显示程序。

8.7　利用图 8.5 电路，编写程序，将 a[4]={1，2，3，4}分别写入 AT24C01，再把这 4 个数读到数组 b[4]中。

第 9 章　单片机应用系统设计与开发

本章主要内容

单片机应用系统设计的一般步骤和方法。单片机应用系统的抗干扰技术。单片机应用系统实例。

9.1　单片机应用系统设计的一般步骤和方法

9.1.1　对单片机应用系统的性能要求

单片机应用系统大多数用于工业环境、嵌入到其他设备或作为部件组装到某种产品中，其基本要求如下：

（1）高可靠性。

（2）较强的环境适应能力。

（3）较好的实时性。

（4）易于操作和维护。

（5）具有一定的可扩充性。

（6）具有通信功能。

总之，单片机应用系统应该能够在恶劣的工业环境中稳定运行，能够对工业控制过程中的突发事件做出快速响应，可靠性和实时性是单片机应用系统的最基本要求。

9.1.2　设计步骤

不同的单片机应用系统，其性能指标、要求、应用场所、难易程度各不相同，因此在整个设计开发过程中，有着不同的步骤和方法，但也存在一些共性。一般来讲可以分为需求分析，总体方案设计、硬件设计与调试、软件设计与调试、系统功能调试与性能测试、产品验收和维护、文件编制和技术归档等。单片机应用系统的开发步骤如图 9.1 所示。

1．需求分析

需求分析就是明确所设计的单片机应用系统要“做什么”和“做的结果怎样”。需求分析阶段的结果是形成可操作的设计需求任务书。主要内容如下：

（1）输入信号：系统所要检测的信号类型、精度要求、信号的变换速率以及信号的量程大小等，以便在方案设计中确定检测元件、检测方法、输入通道的结构和技术等。

（2）输出信号：系统所要输出的信号类型、精度要求、信号制式、功率大小等，以便在方案设计中确定模拟量、开关量输出通道的结构和输出方式。

（3）系统结构：确定是设计的标准产品还是非标产品，是单 CPU 结构还是多 CPU 结构，是否可以构成通信网络，是否挂靠某种网络标准或现场总线标准等。

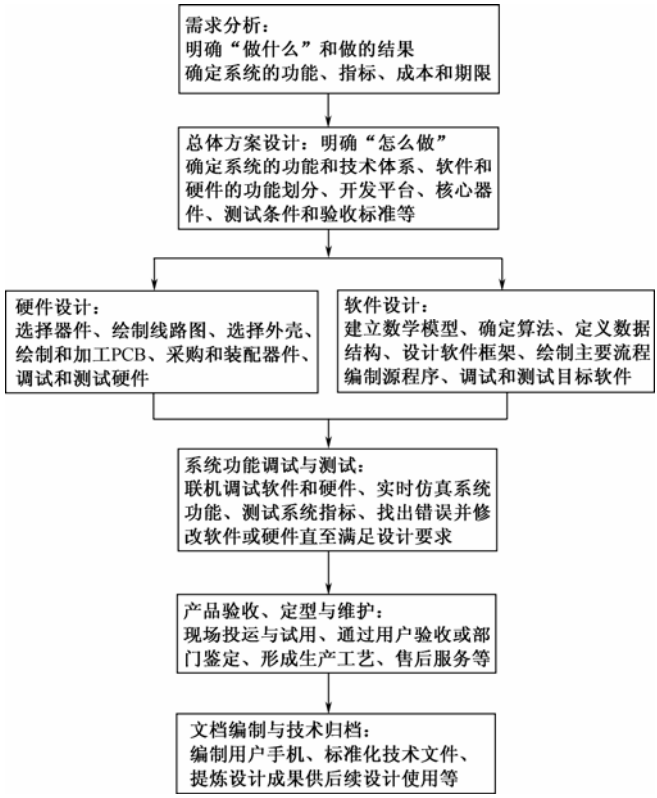


图 9.1　单片机应用系统开发步骤

（4）实时控制及其精度要求：在工业控制领域应用的单片机系统可以称为单片机控制系统，这类控制系统的设计必须考虑实时性需求、生产设备、生产工艺过程和配套电气设备等因素。如果系统具有回路控制功能，应明确系统的控制精度，以便于在方案设计确定所要采用的数字滤波算法、控制算法和各类校正算法等。

（5）系统接口：选择菜单接口还是图形接口，按键输入还是触摸屏输入，以便在方案设计中确定操作方式、参数设置和存储方式、手/自动方式及其切换等。

（6）扩充设计：确定系统是否需要扩充，扩充的形式等。

（7）可靠性设计：确定系统达到何种可靠性指标，以便在方案设计时确定采取何种可靠性设计手段、抗干扰设计技术等。

（8）成本预算：确定完成系统开发所需要的费用，包括硬件费用、软件费用、劳务费、辅料费、税收及其他费用等。

（9）完成期限：如果是用户委托开发或部委科研项目，应确定整个系统的完成期限以及各阶段任务的完成期限，应有具体的时间进度安排。

（10）测试条件和验收标准：应明确系统功能和指标所基于的测试条件，验收系统的具体标准和验收形式。

2．总体设计

系统总体设计解决的是系统采用何种方法，系统结构组成，功能模块具体划分以及模块之间的关系，指标分解等问题。总体设计就是要从宏观上解决“怎么做”。主要内容如下：

（1）确定技术途径：单片机应用系统也有大小之分，不同层次的应用系统可以采用委托

设计、自行设计或二者结合等设计途径。尤其是面向行业领域的单片机应用系统，有必要采取合作开发或有偿使用第三方现有技术的开发形式。

（2）确定技术方法：技术方法是指为实现单片机系统功能目标而准备采用的可行的技术手段，包括高性能器件、合适的开发平台、开发语言、软件算法等。

（3）划分子系统和主要功能块：确定了系统的技术途径和主要技术方法后，就可针对系统的需求，逐步确立系统的体系结构，进行主要功能的划分，确定完成各主要功能的系统，确定各系统的相应功能技术指标和各子系统之间的接口关系。对于每个子系统，又可以进一步划分其硬件功能和软件功能。

（4）确定系统组成框图：用结构框图表示系统（硬件、软件）的体系结构，图中要标明系统各部分的组成结构、各部分之间的接口方式、系统与外界的接口等。

（5）系统的综合与检查：当系统的指标逐项得到分解、各子系统的功能指标均已落实、系统体系已经建立之后，设计者有必要从系统总体角度出发，对各子系统进行综合性分析，检查各模块的功能合成后能否达到系统的功能需求。

系统的总体方案反映了整个系统的综合情况，要从正确性、可行性、先进性、可用性和经济性等角度来评价系统的总体方案。只有当拟定的总体方案能满足上述基本要求后，设计好的目标系统才有可能符合这样的基本要求。总体方案为各子系统的设计与开发提供一个指导性文件。

3．硬件设计

硬件设计的主要内容是基于总体方案选择和采购系统所需的各类元器件，设计系统的电子线路图，选择合适的外观结构，设计和外协加工印制电路板，安装元器件，调试硬件线路等。硬件设计应确保功能设计和接口设计满足系统的需求，并且充分考虑和软件的协调工作关系，注重选用高集成度的器件，充分考虑硬件软化、软件硬化等设计技术。

4．软件设计

软件设计的主要任务是拟定软件设计方案，确立数学模型或算法，划分出主要的软件模块，根据需要绘制部分软件模块的流程图，编制源程序，调试和测试目标软件的基本功能。

5．单片机应用系统的调试

调试是基于系统的设计需求，进行系统功能调试和性能指标的测试，形成测试报告，核对用户需求或设计需求和系统现有功能、指标的一致性程度。如果系统存在局部错误或不能满足某些设计需求，则应提出局部修改意见，并对硬件和软件进行修改，直至满足设计需求。若是方案性的错误或按既定的方案无法满足设计需求，则应考虑修改设计方案。

6．产品验收与维护

单片机应用系统设计或产品开发结束后，必须经过现场投运、试用和用户的验收。属于国家或部委的科研项目，还应通过有关部门的鉴定。产品验收或鉴定结束后，可以实施产品定型，形成生产工艺，进而形成规模化生产。产品正是投放市场后，维护工作就开始了，这步工作一直要持续到该产品退出市场。

7．文件编制与技术归档

为了维护单片机系统，或将目前的设计成果作为资源用于后续设计，编制相应的技术文

档是非常必要的。提供给用户的安装手册、操作手册和维护手册等是技术文档的重要组成部分之一。技术文档必须按国家标准进行编制，经相关人员审核后存入技术档案室进行统一管理。

9.2 单片机应用系统的抗干扰技术

单片机系统都工作于一定的环境中，有时经过千辛万苦开发的单片机应用系统却不能在工业环境中正常工作，其根本原因在于它的工作环境存在干扰。干扰会影响指令的正常运行，从而造成控制事故或控制失灵；在测量通道中产生干扰，就会产生测量误差；计数器受到干扰可能造成计数不准；有时过大的干扰冲击，甚至会造成系统的损坏。

干扰产生于干扰源。干扰可能来至系统内部，也可能来至系统外部。干扰源要形成干扰，还必须有形成干扰的路径或易受干扰的对象，这就是所谓干扰的三要素，即干扰源、干扰路径和易受干扰的系统。干扰源产生的干扰信号是通过一定的耦合通道才对测控系统产生作用的。因此，弄清干扰信号的传递方式十分重要。干扰通过耦合方式影响单片机测控系统，主要有以下几种形式：一是直接耦合方式，干扰信号直接经过线路传导到工作电路中。例如，干扰信号经过电源线进入单片机系统是最常见的直接耦合现象。二是公共阻抗耦合方式，是噪声源与信号源具有公共阻抗时的传导耦合。三是电容耦合方式，电位变化在干扰源与干扰对象之间引起的静电感应，如组件之间、导线之间、导线与组件之间存在的分布电容所引起的噪声传导通路。四是电磁感应耦合方式，交变电流在载流导体周围产生磁场，会对周围的闭合电路产生感应电动势。五是辐射耦合方式，当高频电流流过导体时，在该导体周围便产生高频交变的电力线或磁力线，从而形成电磁波。六是漏电耦合方式，当相邻的组件或导线之间的绝缘阻抗降低时，有些信号便经过绝缘电阻耦合到逻辑组件的输入端形成干扰。

单片机应用系统的抗干扰措施也是针对这三要素进行的，消除干扰源也是最根本的，在这个前提下，下面从硬件和软件两个方面讨论抗干扰问题。

9.2.1 硬件抗干扰技术

1．抑制干扰源常用措施

抑制干扰源首先就是尽量使单片机应用系统远离干扰源，其次是尽可能地减小干扰源的电压变化率和电流变化率，这是抗干扰设计中最优先考虑和最重要的原则。减小干扰源的电压变化率主要是通过在干扰源两端并联电容来实现。减小干扰源的电流变化率则是在干扰源回路串联电感或电阻以及增加续流二极管来实现。

（1）给继电器线圈增加续流二极管，消除断电时产生的反电动势。

（2）在继电器接点两端并接火花抑制电路（一般为 RC 串联电路，电阻一般为几～几十 kΩ，电容为 0.01μF）以减小电火花影响。

（3）给电机加滤波电路，注意电容、电感连线要尽量靠近电机。

（4）电路板上每个 IC 要并接一个 0.01～0.1μF 高频电容，减小 IC 对电源的影响。注意高频电容的布线应靠近电源端，并尽量短，否则等于增大了电容的等效串联电阻，会影响滤波效果。

（5）避免 90° 折线，减小高频噪声发射。

（6）在可控硅两端并接 RC 抑制电路，减小可控硅噪声（这个噪声严重时可能会把可控

硅击穿的）。

2．切断干扰传播途径措施

（1）充分考虑电源对单片机的影响。许多单片机对电源噪声很敏感，要给单片机电源加滤波电路或稳压器，以减小电源噪声对单片机的干扰。

（2）如果单片机的 I/O 口用来控制电机等噪声器件，在 I/O 口与噪声源之间应加隔离，通常采用光电隔离技术。

（3）注意晶振布线。晶振与单片机引脚尽量靠近，用地线把时钟区隔离起来，晶振外壳接地并固定。此措施可解决许多疑难问题。

（4）电路板合理分区，如强、弱信号，数字、模拟信号应分开，尽可能把干扰源 （如电机，继电器）与敏感元件（如单片机）远离。

（5）用地线把数字区与模拟区隔离，数字地与模拟地要分离，最后在一点接于电源地。A/D、D/A 芯片布线也以此为原则。

（6）单片机和大功率器件的地线要单独接地，以减小相互干扰，大功率器件尽可能放在电路板边缘。

（7）在单片机 I/O 口、电源线、电路板连接线等关键地方使用抗干扰元件，如磁珠、磁环、电源滤波器，屏蔽罩，可显著提高电路的抗干扰性能。

3．提高敏感器件的抗干扰性能

提高敏感器件的抗干扰性能是指从敏感器件考虑尽量减小对干扰噪声的拾取，以及从不正常状态尽快恢复的方法。常用措施有：

（1）选择抗干扰性能强的电子元件，对元件进行筛选和老化。

（2）布线时尽量减小回路环的面积，以降低感应噪声。

（3）布线时，电源线和地线要尽量粗。除减小压降外，更重要的是降低耦合噪声。

（4）对于单片机闲置的 I/O 口，不要悬空，要接地或接电源。其他 IC 的闲置端在不改变系统逻辑的情况下接地或接电源。

（5）对单片机使用电源监控及看门狗电路等，可大幅度提高整个电路的抗干扰性能。

（6）在速度能满足要求的前提下，尽量降低单片机的晶振和选用低速数字电路。

（7）IC 器件尽量直接焊在电路板上，少用 IC 座。

（8）有条件采用四层以上印制板，中间两层为电源和地。

9.2.2　软件抗干扰技术

软件抗干扰技术是当系统受干扰后，使系统恢复正常运行或输入信号受干扰后去伪存真的一种辅助方法。此技术属于一种被动抗干扰措施，但是由于软件抗干扰设计灵活，节省硬件资源，操作起来方便易行，所以软件抗干扰技术越来越受到人们的重视。

1．数字滤波技术

一般在模拟量输入信号中，均含有噪声和干扰。为了进行准确的测量和控制，必须消除或最大程度地降低噪声和干扰的影响。常用的办法有硬件滤波电路和软件滤波程序。

（1）数字滤波的优点。数字滤波器实际上是一段程序，它的主要优点在于：

① 数字滤波器是用程序实现的，不需要增加硬设备，所以可靠性高，稳定性好。

② 数字滤波器可以对频率很低的信号实现滤波，克服了模拟滤波器的缺陷。

③ 数字滤波器可以根据信号不同，采用不同的滤波方法或滤波参数，具有灵活、方便、功能强的特点。

（2）常用的数字滤波方法。

① 算术平均值法。算术平均值法是求 N 个采样数据的算术平均值，这样得到的结果与采样值的偏差的平方和最小。这种方法适合于一般的随机干扰信号的滤波。N 值的选择应根据具体的滤波对象而定，N 值大，则平滑度高，但灵敏度低；N 值小，则平滑度低，但灵敏度高。

② 中位值滤波法。中位值滤波法的原理是对被测信号连续采样 M 次（$M \geq 3$）且是奇数，并按大小顺序排序，取其中间值作为本次的有效值。

③ 限幅滤波法。由于大的随机干扰或采样器不稳定，使得采样数据偏离实际值太远，在这种情况下，当信号大于某设定值时，就取设定值为采样值；当信号小于某设定值时，就取设定值为采样值。有时根据信号变化率来决定是否采取限幅滤波。

④ 惯性滤波法。惯性滤波法实际上与硬件的一阶惯性滤波器（RC 低通滤波器）的功能一样，只是滤波系数可以随意调整，使用灵活。

2．指令冗余技术

当 CPU 受到干扰后，往往将一些操作数当作操作码来执行，引起程序混乱。这时，我们首先要尽快将程序纳入正轨。MCS-51 单片机指令系统中所有的指令都不超过 3 字节，而且有很多单字节指令。当程序弹飞到某一单字节指令上时，便自动纳入正轨。当弹飞到某一双字节或三字节指令上时，有可能落在其操作数上，从而继续出错。因此，在程序当中应多用单字节指令，并在关键地方有意识地安排一些 NOP 指令，或将有效单字节指令重复书写，这就是指令冗余。指令冗余会降低系统的效率，但一般情况下 CPU 有足够的时间来执行这些冗余的指令。

在双字节和三字节指令之后插入两条 NOP 指令，可保证其后的指令不被拆散。或者说在某指令前插入两条 NOP 指令，则这条指令就不会被前面执行下来的失控程序拆散，并将完整执行，从而使程序走上正轨。但不能在程序中插入太多的冗余指令，以免明显降低程序的运行效率。因此，在一些对程序流向起决定作用的指令之前插入两条 NOP 指令，以保证弹飞的程序能尽快纳入正确的控制轨道。此类指令有：RET、RETI、ACALL、LCALL、SJMP、AJMP、LJMP、JZ、JNZ、JC、JNC、JB、JNB、JBC、CJNE、DJNZ 等。在某些对系统工作状态至关重要的指令（如 SETB EA 之类）前也可插入两条 NOP 指令，以保证被正确执行。

3．软件陷阱技术

指令冗余使弹飞的程序安定下来是有条件的，首先弹飞的程序必须落在程序区，其次必须执行到冗余指令。当弹飞的程序落到非程序区（如 ROM 空间未使用部分，数据表格区）时，则第一个条件不满足。当弹飞的程序在没有碰到冗余指令之前，已经自动形成一个死循环，这时第二个条件也不满足。对于不满足第一个条件可以采取软件陷阱措施来解决，对于不满足第二条件可以采取“看门狗”电路来解决。

所谓软件陷阱，就是一条引导指令，强行将捕获的程序引向一个指定地址，在那里有一段专门对程序出错进行处理的程序。如果把这段程序的入口标号称为 ERR，则软件陷阱即为

一条 LJMP ERR 指令。为加强捕获效果，一般还在它前面加两条 NOP 指令。其构成如下：

```
NOP
NOP
LJMP   ERR
```

软件陷阱可以安排在未使用 ROM 空间或程序当中。没有使用的 ROM 空间（如未使用的中断向量区、程序为占用的大片空间）用软件陷阱去填充，只要程序飞到这些空间里都可以得到处理。在数据区或表格的中间不安排软件陷阱，以免破坏数据或表格的连续性和一一对应关系，而只能在表格的末尾安排软件陷阱。如果程序飞到表格中间，则只能寄希望其他软件陷阱和冗余指令来制服它了。在程序区，软件陷阱只能安排在程序断点之后，正常程序永远也执行不到的地方，如 LJMP、SJMP、AJMP、RET、RETI 的指令后面，如果执行了软件陷阱，则表明程序已经出错。如下所示：

```
        ……
        AJMP   L1
        NOP
        NOP
        LJMP   ERR
        ……
        AJMP   L2
        NOP
        NOP
        LJMP   ERR
L1：    ……
L2：    ……
ERR：       ……
```

4．输入口信号重复检测

对于重要开关量输入信号的检测，实际应用中一般采用 3 次或 5 次重复检测的方法，即对接口中的输人数据信息进行重复进行 3 次或 5 次检测，若结果完全一致则认为是“真”的输入信号，若多次测试结果不一致，既可以停止检测显示故障信息，又可以重复进行再检测。

对于软件测量而言，输入量干扰大多数是叠加到有效信号上的一系列作用时间短的尖脉冲，但是频率不一致，因此应在相邻的检测之间应有一定的时间间隔。理论上可以是等时间段的，而在实际使用过程中，由于外部环境比较复杂，等时间段只能滤除某个频段的干扰，为了滤除尽可能多的干扰，间隔时间应为不等的时间段，但是对数据影响较大的尖峰，其作用的时间宽度在几十到几百μs 之间，所以把滤波时间限定在 ms 级上。需要注意的是，对于软件时序要求比较严格场合，延时查询时间不宜过长，查询次数一般以 3 次为宜。

5．输出端口数据刷新

开关量输出抗干扰技术主要采用的方法是重复输出，即同一指令连续执行多次，这是提高输出信号稳定性的有效措施之一。

6．程序自检

程序自检是提高测控软件可靠性的有效方法之一。自检程序主要是对单片机系统的主要

器件如单片机的 I/O 口、外部扩展的可编程 I/O 接口芯片、A/D 器件、ROM 器件等进行检测，如出现故障能够给出故障提示。因此自检程序不但可以了解与测试相关外设的工作情况，而且可避免因外设原因而使测控系统不能正常工作的干扰。

9.3 简易电脑时钟设计

9.3.1 功能要求

设计并实现简易电脑时钟系统，要求用尽可能少的常用电子器件实现以下功能：

（1）实时显示当前时间即北京时间（时、分）。

（2）可修改北京时间（时、分）。

（3）具有闹钟功能，闹钟时的音乐为“两只老虎”。

（4）可设定闹铃时间（时、分）。

（5）具有闹钟贪睡功能。

9.3.2 总体设计

根据系统功能要求，可将系统组成结构分成四大部分：单片机控制中心、键盘接口、时钟显示和声音报时，系统的组成结构如图 9.2 所示。其中，单片机控制中心是核心。MCU 根据按键输入，可切换不同的显示模式或设置不同的参数。时钟显示完成北京时间或秒表时钟的信息。声音报时可完成闹铃的提示。

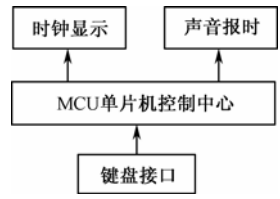


图 9.2 电脑时钟组成结构图

（1）时钟显示。选取 4 位 LED 数码管，用于显示小时和分钟。

（2）声音报时。选用最常见的也是最简单的声音提示方式——蜂鸣器。

（3）键盘接口。设置三个按键，即模式（MODE）键、UP 键和 DOWN 键。按键用来设置北京时间、闹钟时间和启动秒表等功能。

9.3.3 硬件设计

1．微处理器选择

遵循成本低、性能高、满足功能要求等原则选择微处理器。在本系统中，采用 4 位 LED 显示，动态扫描 4 位 LED 数码管进行数字显示，需要 12 位 I/O 线，声音提示需要 1 位 I/O 线，按键需 3 位 I/O 线，因此，系统接口需要至少 16 位单片机 I/O 线。而普通 MCS-51 单片机一般有 32 位 I/O 线，选用普通单片机即可满足要求。考虑到 STC89C51 微控制器具有价格非常低廉、程序空间大、资源较丰富、在线下载非常方便等特点，并且该芯片功能与普通 51 芯片相同，编程方面亦可采用所熟悉的 Keil 软件，因此选择 STC89C51 微控制器。

2．电路设计

电路设计包括电路原理图设计和印制电路图设计，电路原理图如图 9.3 所示。由于系统的工作电源为 5V，为了使用方便，用 USB 接口供电，即系统直接从 USB 接口获取 5V 电源。

系统电路图如图 9.3 所示。

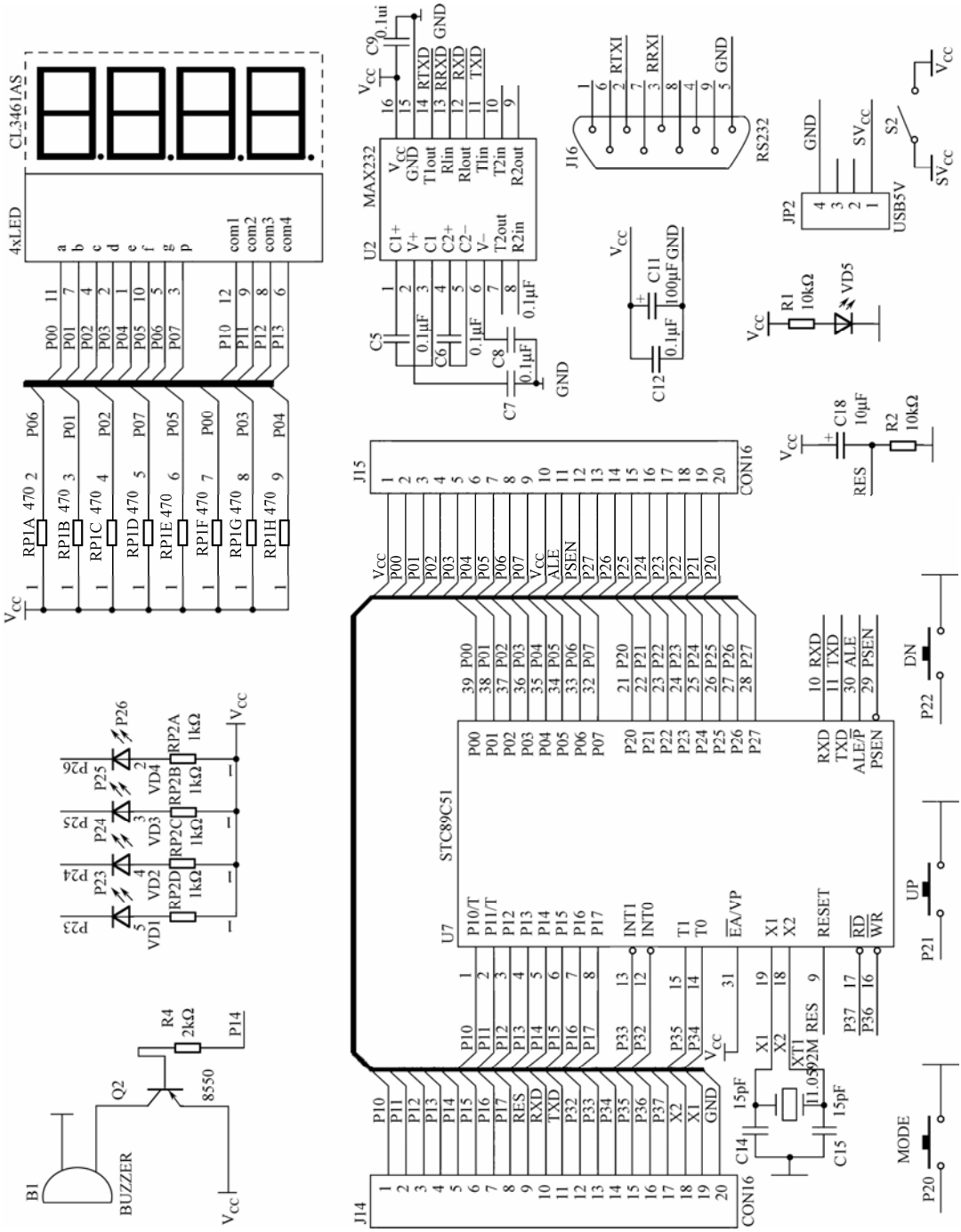


图 9.3　系统电路原理图

9.3.4　软件设计

1．操作功能设计

系统设有四位数码管，三个按键。采用菜单式人机对话，便于操作。系统分成多种菜单，

MODE 键切换菜单，UP 或 DOWN 键设置相关参数。菜单设计及操作如下：

（1）北京时间显示菜单：数码管前两位显示小时，后两位显示分钟，中间小数点 0.5s 亮，0.5s 灭。

（2）按 MODE 键一次，进入修改小时模式，第一个指示灯 D1 亮，按 UP 或 DOWN 键可对小时进行修改（加或减）。

（3）按 MODE 键二次，进入修改分钟模式，第二个指示灯 D2 亮，按 UP 或 DOWN 键可对分钟进行修改（加或减）。

（4）按 MODE 键三次，进入修改闹钟小时模式，第三个指示灯 D3 亮，按 UP 或 DOWN 键可对闹钟小时值进行修改（加或减）。

（5）按 MODE 键四次，进入修改闹钟分钟模式，第四个指示灯 D4 亮，按 UP 或 DOWN 键可对闹钟分钟值进行修改（加或减）。

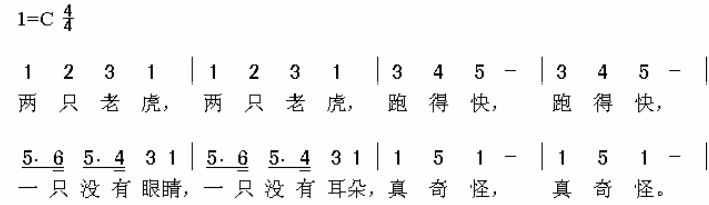
（6）按 MODE 键五次，数码管上显示“LL.00”或“LL.01”，“LL.00”表示贪睡闹铃功能关闭，01 表示贪睡闹铃功能开启，按 UP 或 DOWN 键可在 00 或 01 间循环选择。在贪睡闹铃功能关闭时，可按任意键停止当前闹钟，或在无按键情况下会闹钟 1 分钟，以后不再响。在贪睡闹铃功能开启时，按下除 UP 键以外的任意键，则停止当前闹钟；或在无按键情况下会闹钟 1 分钟。每 5 分钟，闹钟会重新响起，如此循环。只有按下 UP 键时，才能停止闹钟以后再不响。

（7）按 MODE 键六次，返回到北京时间显示模式。

2．编程思路

使用定时器 0 完成数码管扫描、按键扫描、时钟的计时功能。定时器 0 设为 16 位计时模式，定时 5ms，即每 5ms 扫描一位数码管，20ms 循环一次。按键扫描过程中，需进行去抖动处理。扫描到相应的按键时，执行相应的操作，如扫描到 MODE 键时，进行菜单切换操作。切换菜单的方法是通过程序定义一个变量，根据变量不同的取值来表示不同的菜单。在按键处理后，及时更新数码管显示的内容。

使用定时器 1 完成音乐播放功能。“两只老虎”歌谱如下：



3．程序源代码

程序源代码如下：

```
#include "reg52.h"
#include <intrins.h>
#define THCO     0xee
#define TLCO     0x0
#define    FREQ 11059200
unsigned char    code Duan[]={0x3F, 0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x38};
```

```
//共阴极数码管，0-9 段码表
unsigned char    Data_Buffer[4]={0,0,0,0}；      //四个数码管显示数值，数组变量定义
unsigned char Hour=0,Min=0,Sec=0；               //实时时间
unsigned char N_Hour=0,N_Min=0；                 //闹钟时间
bit N_flag=0；                                   //1\0：贪睡功能开启或关闭
bit Brush_flag=0；                               //更新数码管标志
sbit P10=P1^0；                                  //四个数码管的位码口定义
sbit P11=P1^1；
sbit P12=P1^2；
sbit P13=P1^3；
sbit K_Mode=P2^0；                               //按键定义
sbit K_UP=P2^1；
sbit K_DN=P2^2；
sbit LED1=P2^3；                                 //四个指示灯
sbit LED2=P2^4；
sbit LED3=P2^5；
sbit LED4=P2^6；
unsigned char Mode=0；                           //MODE，0：实时时钟；1：修改小时；2：修改分钟，
                                                 3：修改闹钟小时，4：修改闹钟分钟，5：启停闹钟
bit N_Ring=0；                                   //启动闹铃标志，1：启动
bit N_keyclose=0；                               //1：贪睡功能开启时，有键按下，即关闭贪睡闹钟

unsigned char code music_tab[]={                 //两只老虎乐谱，高 4 位为音乐节拍，低 4 位为音符
        0x31,0x32,0x33,0x31,0x31,0x32,0x33,0x31,0x33,0x34,0x45,0x33,0x34,0x45,
        0x25,0x16,0x25,0x14,0x33,0x31,0x25,0x16,0x25,0x14,0x33,0x31,
        0x31,0x35,0x41,0x31,0x35,0x41,0 };
unsigned char code music_l_tab[8]={0,1,2,3,4,6,8,16}；  //节拍延时单位
unsigned char code music_freq_tab[16]={
         0xff,0xea,                                  //0  休止符
         255-FREQ/24/1049/256,                       //1 do  高字节  //255-FREQ/24/x/256
         255-FREQ/24/1049%256,                       //1 do  低字节  //256-FREQ/24/x%256
         255-FREQ/24/1178/256,256-FREQ/24/1178%256,        //2 re
         255-FREQ/24/1322/256,256-FREQ/24/1322%256,        //3 mi
         255-FREQ/24/1400/256,256-FREQ/24/1400%256,        //4 fa
         255-FREQ/24/1572/256,256-FREQ/24/1572%256,        //5 suo
         255-FREQ/24/1665/256,256-FREQ/24/1665%256,        //6 la
         255-FREQ/24/1869/256,256-FREQ/24/1869%256,        //7 xi
 };
unsigned char temp_TH1；
unsigned char temp_TL1；
sbit PIN_MSC=P1^4；                              //音乐输出端口
```

```
void music_delay(unsigned char x);              //音乐节拍延时
void music_play(unsigned char *msc);            //播放音乐子程序
void music_int_t1 (void) interrupt 3 using 1    //定时中断 1
  {
                PIN_MSC=~PIN_MSC;
                TH1=temp_TH1;
                TL1=temp_TL1;
  }
void music_delay(unsigned char n)               //延时 125×n 毫秒
  {
                unsigned char i=125,j;
                do {
                    do {
                        for (j=0;  j<230;  j++) _nop_();
                    }while(--i);
                }while(--n);
  }
void music_play(unsigned char *msc)                 //音乐
  {
                unsigned char music_long;       //节拍
                unsigned char music_data=0;     //音符数据
                temp_TH1=0xff;
                temp_TL1=0xea;                  //关输出（输出超声波）
                TH1=temp_TH1;
                TL1=temp_TL1;
               TR1=1;                           //开 T1 定时器中断
                while (*msc != 0x00&&N_Ring==1)         //闹铃一直启动
                    {
                    music_data=*msc & 0x07;
                    music_long=*msc>>4;
                    if (music_long != 0)        //是音符
                        {
                        temp_TH1=music_freq_tab[music_data*2];
                        temp_TL1=music_freq_tab[music_data*2+1];
                     music_delay(music_l_tab[music_long&0x07]);
                         }
                    msc++;    }
                TR1=0;                          //播放结束
                PIN_MSC=1;                      //关输出
}
void main()
```

```
{
  TMOD=0x11;                          //定时器 0 初始化
  TH0=THCO;
  TL0=TLCO;
  TR0=1;
  ET0=1;
  EA=1;
  TR1=0;
  ET1=1;
  while(1)
  {
     if(N_Ring==1)                    //闹铃启动
      {
            music_delay(10);
            music_play(music_tab);    //播放音乐一次
       }
    if(Brush_flag==1)                 //更新数码管显示
     {
      Brush_flag=0;
      if(Mode<=2)                     //实时时间
       { Data_Buffer[0]=Hour/10;
         Data_Buffer[1]=Hour%10;
         Data_Buffer[2]=Min/10;
         Data_Buffer[3]=Min%10; }
      else if(Mode<=4)                //闹钟时间
       { Data_Buffer[0]=N_Hour/10;
         Data_Buffer[1]=N_Hour%10;
         Data_Buffer[2]=N_Min/10;
         Data_Buffer[3]=N_Min%10; }
      else if(Mode==5)
      { Data_Buffer[0]=10;            //L 段码序号
         Data_Buffer[1]=10;
         Data_Buffer[2]=0;
         Data_Buffer[3]=N_flag; }
    }
  }
}
void timer0() interrupt 1
{
 static unsigned char Bit=0;          //静态变量，退出程序后，值保留
 static unsigned char count=0,K_count=0;
```

```
 static unsigned char time_n=0;
TH0=THCO;
 TL0=TLCO;
count++;
 if(count>=200)                                  //秒计时，定时器定时 5ms，计 200 次为一秒
 {
   count=0;
   Sec++;
   if(Sec>=60)
   {
       Sec=0;
       Min++;       Brush_flag=1;              //更新数码管显示
       if(Min>=60)                             //时间计时
            {
               Min=0;   Hour++;
               if(Hour>=24)Hour=0;
            }
      if(N_Hour==Hour&&N_Min==Min)             //闹钟相等
              {N_Ring=1；N_keyclose=1；}
      else N_Ring=0;                           //贪睡功能关闭，则最多闹一分钟
      if(N_keyclose&&N_flag)                   //贪睡功能开启，且未按键关闭
      {
              if(time_n==0)N_Ring=1;           //第一次需开启闹钟
          time_n++;
          if(time_n>=5){time_n=0；}            //后 4 分钟关闭闹钟
      }
      else time_n=0;
   }
 }
Bit++;
 if(Bit>=4)Bit=0;
 P1|=0x0f;                                     //先关位码
 P0=Duan[Data_Buffer[Bit]];                    //开段码
 if(count<100&&Bit==1)P0|=0x80;                //0.5S 中间小数点亮，之后灭，不断循环
 switch(Bit)                                   //送位码
 {
  case 0: P10=0；break;
  case 1: P11=0；break;
  case 2: P12=0；break;
  case 3: P13=0；break;
 }
```

```
if(K_Mode==0||K_UP==0||K_DN==0)                    //有键按下
 {
  K_count++;
      if(K_count>=30)                              //消抖处理
      {
     if(N_flag==0&&N_Ring==1)                      //贪睡功能关闭,且闹铃启动
        {N_Ring=0；return；}
      K_count=0；
      Brush_flag=1；
      if(K_Mode==0)                                //修改 Mode
           {
            Mode++；
            if(Mode>=6)Mode=0；
            if(Mode==0){LED1=1；LED2=1；LED3=1；LED4=1；}     //实时时间
            else if(Mode==1)LED1=0；                          //修改小时
            else if(Mode==2){LED1=1；LED2=0；}                //修改分钟
            else if(Mode==3){LED2=1；LED3=0；}                //修改分钟
            else if(Mode==4){LED3=1；LED4=0；}                //修改分钟
           }
       else if(K_UP==0)
           {
            if(N_Ring==1&&N_flag==1)
                {N_Ring=0；N_keyclose=0；return；}            //关闭闹铃
            if(Mode==1)                                       //实时时间小时加
                { Hour++；  if(Hour>=24)Hour=0；}
            else if(Mode==2)                                  //实时时间分钟加
                { Min++；  if(Min>=60)Min=0；  }
            else if(Mode==3)                                  //闹钟时间小时加
                { N_Hour++；  if(N_Hour>=24)N_Hour=0；}
            else if(Mode==4)                                  //闹钟时间分钟加
                { N_Min++；  if(N_Min>=60)N_Min=0；}
            else if(Mode==5)      N_flag=!N_flag；
           }
       else if(K_DN==0)
           {
            if(Mode==1)                                       //小时减
              {if(Hour==0)Hour=23；
                else Hour--；}
            else if(Mode==2)                                  //分钟减
              { if(Min==0)Min=59；
                else Min--；}
```

```
                else if(Mode==3)                                    //小时减
                  {if(N_Hour==0)N_Hour=23;
                    else N_Hour--;  }
                else if(Mode==4)                                    //分钟减
                  {if(N_Min==0)N_Min=59;
                    else N_Min--;}
                else if(Mode==5)      N_flag=!N_flag;
               }
    }
  }
  else K_count=0;
}
```

9.4 数字电压表设计

9.4.1 功能要求

数字电压表（Digital Voltmeter，DVM）是采用数字化测量技术，把连续的模拟电压量转换成离散的数字化形式并加以显示的仪表。设计一数字电压表，要求用专用 A/D 转换器来实现 8 路电压采集，具体参数要求如下：

（1）输入 8 路直流电压值，输入范围为 0～5V。

（2）测量精度能够达到 0.02V。

（3）4 位 LED 数码管轮流显示 8 路测量结果。

9.4.2 总体设计

根据系统要求，单片机主要完成通道的选择、电压值的采集、以及数字显示等功能。系统由主控电路、显示电路、A/D 转换电路、电源电路及复位电路等部分组成，系统组成框图如图 9.4 所示。

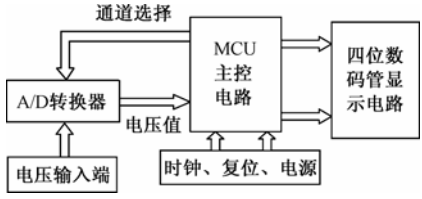


图 9.4　数字电压表系统组成框图

9.4.3 硬件设计

1．器件选择

A/D 转换器选择：由于输入的直流电压信号有 8 路，且测量精度为 0.02V，所以选择 ADC0809 作为 A/D 转换器完全满足要求。另外，要求输入 8 路电压信号，选用 ADC0809 芯片可以完成 A/D 转换要求。

单片机选择：ADC0809 与单片机之间连接时，需要单片机 15 根 I/O 线。4 位 LED 显示和 LED 的位选一共需要 12 根 I/O 线。普通 51 系列单片机均可满足要求。在此选择 AT89C52 微控制器。

2．原理图设计

数字电压表原理图如图 9.5 所示。引脚安排如下：P0、P2.0～P2.3 口作为数码管显示控制，P1 口作 A/D 转换数据输入，P3.0～P3.6 口用作 ADC0809 的控制。其中，ADC0809 的 CLK 可以通过单片机 ALE 脚的六分频再通过 74LS74 构成的四分频得到 500kHz 时钟信号得到。

3．Proteus与Keil仿真平台建立

作为功能强大的 EDA 工具软件，Proteus 是模拟单片机及其外围器件的很好工具，而 Kei1 软件是目前广为应用的 51 系列单片机软件开发工具之一。将它们完美结合，可为单片机系统资源、软件技术、硬件接口电路、软件和硬件相结合的应用系统提供一个很好的理论和实践相结合的实验仿真平台。Proteus 与 Keil 的整合过程如下：

（1）安装 Proteus6.7SP3 和 Keil uVision3 软件。

（2）把 Proteus 安装目录下 VDM51.dll(C:\Program Files\Labcenter Electronics\Proteus6 Professional\MODELS\)文件复制到 Keil 安装目录下的\C51\BIN 目录中。

（3）编辑 Keil 目录下 tools.ini 文件，在[C51]项下，加入：TDRV5=BIN\VDM51.DLL ("PROTEUS MONITOR-51 DRIVER")（注意，TDRVX 为单片机软件仿真的硬件配置设置）。

（4）打开 Proteus ISIS 软件。设计单片机实验硬件组成框图，建立硬件连接原理图，如图 9.5 所示。

（5）打开 Keil uVison3 软件。建立单片机系统软件工程，针对实验要求，编制程序。

（6）在 Keil uVison3 软件中，选择菜单“Project”->“Options for Target‘Target1’”，在出现的对话框中，选择“Output”页面，选中“Creat Hex File”选项；选择“Debug”页面，选中“Use”-PROTEUS VSM MONITOR-51 DRIVER，进入“Settings”，Host 设为 127.0.0.1，Port 设为 8000。

（7）在 Proteus ISIS 软件中，选择菜单“Source”->“Add/Remove Source Code Files”，在出现的对话框中点击“Chang”按钮，选择从 Keil uVison3 软件工程中所生成的 HEX 文件，点击“OK”。在 Debug 菜单下选中“Use Remote Debug Monitor”。

（8）在 Keil uVison3 软件中，直接进行仿真，连续运行或单步运行，即可在 Proteus ISIS 软件中看到单片机硬件仿真运行结果。

在整合过程中，单片机仿真平台建立过程为（1）～（3）项，实验过程为（4）～（8）项。

9.4.4 软件设计

系统设有四个数码管，要求能够轮流显示 8 路电压输入值。电压精度要达到 0.02V，因此，电压显示范围应为 0.00V～5.00V，用后三位数码管可完成电压显示，第一位数码管可显示对应的通道。

程序编写的思路为：使用定时器 0 完成数码管扫描功能。主程序每 1s 切换一次通道并进行电压数据采集和更新显示。参考源程序如下：

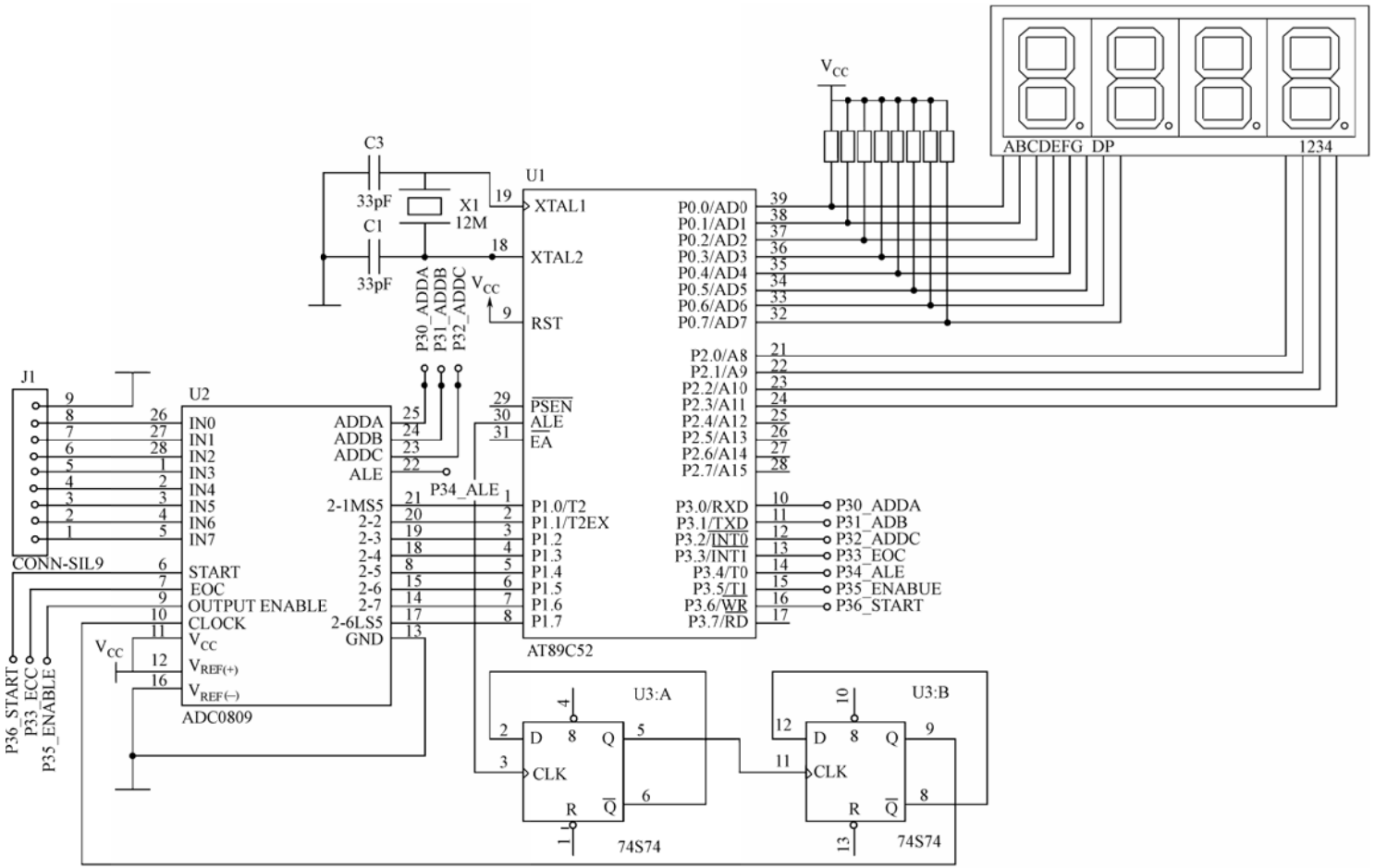


图 9.5　数字电压表原理图

```
#include "reg51.h"                          //添加头文件
unsigned char duan_table[]=                 //段码表
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
unsigned char disp_buf[4]={0,0,0,0,};       //显示缓冲区
sbit ADC0809_EOC=P3^3;
sbit ADC0809_ALE=P3^4;
sbit ADC0809_OE=P3^5;
sbit ADC0809_ST=P3^6;
unsigned char channel=0;
unsigned int V_value=0;                     //电压值单位为 10mV
unsigned int count=200;                     //1s 切换一次通道,计时变量
void AD0809_Sample()
{
P3=(P3&0xf8)|channel;                       //选择通道
ADC0809_ALE=1;
ADC0809_ST=0;
ADC0809_ST=1;
ADC0809_ST=0;
while(ADC0809_EOC==0);                      //烦闷 AD 是否转换结束
ADC0809_OE=1;
V_value=P1;
V_value=V_value*500/256;
ADC0809_OE=0;
}
void timer0() interrupt 1                   //定时器 0 中断子程序
{
    static unsigned char Bit=0;
  TH0=0xee;
  TL0=0;
  if(count!=0)
  count--;
  P2 =P2|0x3f;                              //关位码
  P0= duan_table[disp_buf[Bit]];            //段选
  if(Bit==0||Bit==1)P0|=0x80;
  switch(Bit)                               //开位码
  {
      case 0:   P2&=0xfe;  break;
      case 1:   P2&=0xfd;  break;
      case 2:   P2&=0xfb;  break;
          case 3:   P2&=0xf7;  break;
   }
```

```
    Bit+=1；if(Bit>=3)    Bit=0；
}
main()     //主程序
{
   ADC0809_ST=0；//START=0；
   ADC0809_OE=0；//OE=0；
   TMOD=0x11；
   TH0=0xee；                          //定时器 0 5ms
   TL0=0；
   ET0=1；                             //使能定时器 0 中断
   TR0=1；
   EA=1；
   while(1)
   {
          disp_buf[0]=channel；
     disp_buf[1]=V_value/100；
     disp_buf[2]=V_value/10%10；
     disp_buf[3]=V_value%10；

       while(count!=0)；                 //1s 采集一次
    count=200；                          //1s 重新计数
    channel++；
    if(channel>=8)channel=0；
    AD0809_Sample()；
   }
}
```

9.5　电动小车动态平衡系统

9.5.1　功能要求

设计并制作一个电动车跷跷板，在跷跷板起始端 A 一侧装有可移动的配重。配重的位置可以在从始端开始的 200mm～600mm 范围内调整，调整步长不大于 50mm；配重可拆卸。电动车从起始端 A 出发，可以自动在跷跷板上行驶。电动车跷跷板起始状态和平衡状态示意图分别如图 9.6 和图 9.7 所示。

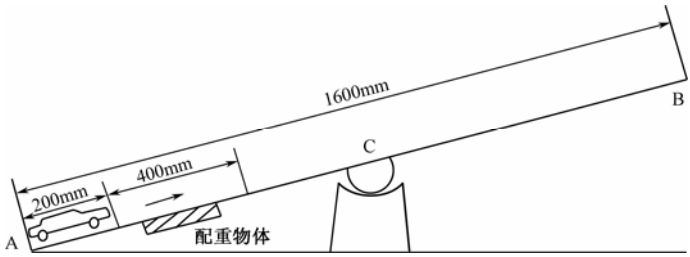


图 9.6　起始状态示意图

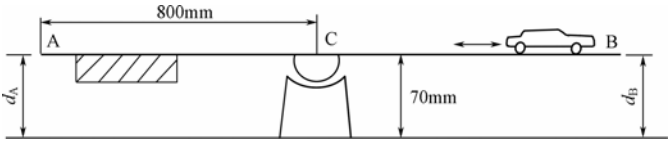


图 9.7　平衡状态示意图

在不加配重的情况下，电动车完成以下运动：

（1）电动车从起始端 A 出发，在 30 秒钟内行驶到中心点 C 附近。

（2）60 秒钟之内，电动车在中心点 C 附近使跷跷板处于平衡状态，保持平衡 5 秒钟，并给出明显的平衡指示。

（3）电动车从平衡点出发，30 秒钟内行驶到跷跷板末端 B 处（车头距跷跷板末端 B 不大于 50mm）。

（4）电动车在 B 点停止 5 秒后，1 分钟内倒退回起始端 A，完成整个行程。

（5）在整个行驶过程中，电动车始终在跷跷板上，并分阶段实时显示电动车行驶所用的时间。

将配重固定在可调整范围内任一指定位置，电动车完成以下运动：

（1）将电动车放置在地面距离跷跷板起始端 A 点 300mm 以外、90° 扇形区域内某一指定位置（车头朝向跷跷板），电动车能够自动驶上跷跷板，如图 9.8 所示。

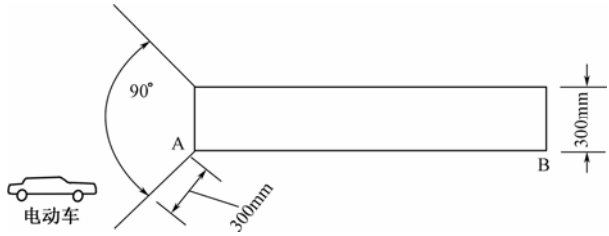


图 9.8　自动驶上跷跷板示意图

（2）电动车在跷跷板上取得平衡，给出明显的平衡指示，保持平衡 5 秒钟以上。

（3）将另一块质量为电动车质量 10%～20%的块状配重放置在 A 至 C 间指定的位置，电动车能够重新取得平衡，给出明显的平衡指示，保持平衡 5 秒钟以上。

（4）电动车在 3 分钟之内完成全过程。

说明：平衡的定义为 A、B 两端与地面的距离差 $d=|d_A-d_B|$不大于 40mm。

9.5.2　总体设计

根据题目要求，对系统关键点进行以下分析和设计。

1．车型选择

电动车的车体可选用市场常见的物美价廉的玩具车。但在跷跷板上行驶存在上坡和下坡的难度，即上坡需要一定的马力和起始速度，下坡需考虑自身惯性导致速度难以控制而驶出跷跷板 B 端外的情况，而这些起始驱动力及惯性因素为不定因素，从而导致速度控制的难度也是不可预知的。因此，选用坦克型车体。存在两种理由：一方面，由于坦克型车体本身的特点，电机驱动方式不同一般车型，它行驶速度均匀，即使在坡度 45° 角的上坡也能均速行驶；另一方面，坦克的马力强大，即使有再多的故障，它也能直接压过，爬坡对于坦克车体

来说是一件非常容易的事情。这样大大降低了电动车的控制难度。

2．车体行驶设计

电动车只能在跷跷板上行驶，为避免驶出跷跷板外，需进行轨迹设计，即为电动车指引行驶的线路。检轨迹线的传感器可选用一对反射式红外传感器，正确定位传感器是寻迹的关键。为了使电动车能沿跷跷板前进和后退，为使电动车准确地行驶到 B 端和准确地退回到 A 端，在电动车的前后各安装 3 对红外对管（如图 9.9），且跷跷板上绘制轨迹图（如图 9.10）。当驶到 B 端时，只有前面的 3 对红外管可同时检测到轨迹，当驶回 A 端时，只有后面的 3 对红外管均可同时检测到轨迹，当在 A 到 B 的中间行驶时，可根据前端中的红外管进行行驶方向判断，在 B 到 A 的中间行驶时（即倒行时），可根据后面中的红外管进行行驶方向判断。这样设计轨迹线也同样可以保证车体的准确停止定位，而系统本身没有对距离进行显示要求，这样也可省去距离检测功能，降低了系统设计难度。

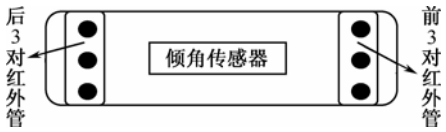


图 9.9　传感器安装示意图

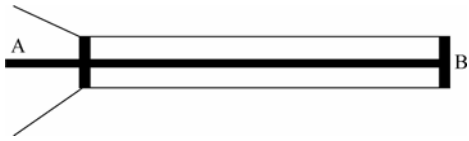


图 9.10　板上行驶轨迹示意图

3．系统平衡设计

小车从开始上坡到达到 B 点，跷跷板的水平角度从 arcsin（140/1600）变为−arcsin（140/1600），绝对平衡时的水平角度为 0°，即处于图 9.7 所示状态。但要使车体保持在绝对平衡的状态是非常困难的。系统要求的平衡状态是在一定范围内，即$|d_A-d_B|$<=40mm。所以，平衡控制的角度范围为：arcsin（40/1600）至−arcsin（40/1600）。为检测该水平角度，需选用单轴倾角传感器，且该传感器安装在车体上最为方便（如图 9.9）。

4．其他功能设计

系统需要显示各阶段的时间，在平衡时作出平衡指示，因此控制系统要有显示、声光报警电路。电动车在行驶过程中，要按方向来行驶，因此存在电机驱动电路。电动车行驶及控制系统均需要不同的电源，在锂电池供电的情况下，需要设计独立供电电路等等。

总上分析，电动车跷跷板控制系统主要有电源供应模块，轨迹探测模块、倾角检测模块、显示模块、电机驱动模块、声光提示模块等组成，如图 9.11 所示。

9.5.3　硬件设计

1．独立供电模块

小车由 7.4 伏的锂电池供电。锂电池在刚充满电时，电压高达 8.4V，随着小车的工作时间的增加，电池的电压会不断的降低到 6V，并且电机及控制电路均需要稳定的直流 5V 电源。因此需要一定的措施才能把锂电池的不稳定电压变换成稳定的 5V。

使用独立供电技术，把电机和控制电路的稳压供电分成两部分电路来实现。考虑到电机的电流比较大，如果采用常用的 7805 类三端线性稳压芯片来实现的话，效率会很低，发热会比较严重。所以使用新型的 LM2596 集成 DC/DC 的开关稳压电路专门给电机提供稳定驱动电源，效率高达 90%，温升小于 10℃。而控制系统的工作电流较小，但要求纹波非常低，所

以使用线性低压差稳压器（LDO）SP1117 单独给控制系统供电，电机与控制系统之间用光电隔离，彻底解决由于电池电量的变化而影响电机速度控制的问题；并且此方案会使控制系统不受电机运行的电气噪声干扰，为系统的稳定工作奠定了基础。LM2596 具体电路及 SP1117 电路图如图 9.12 所示。

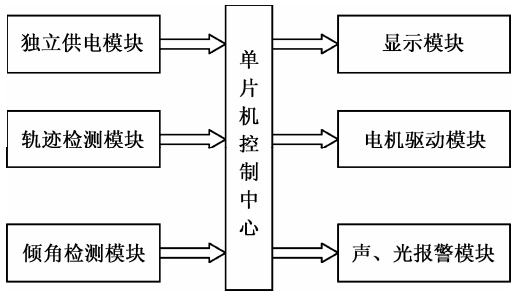


图 9.11　系统组成框图

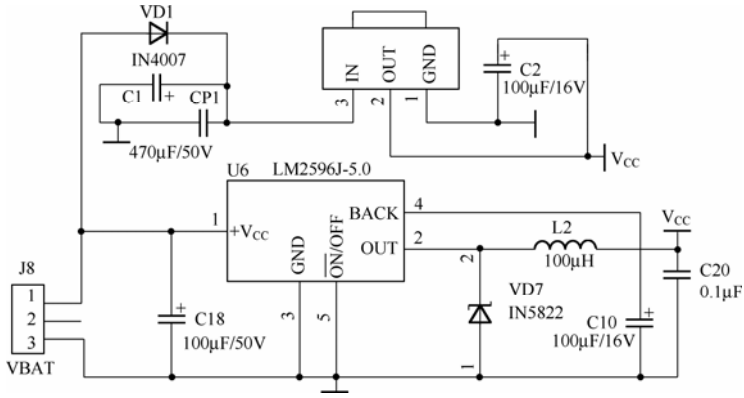


图 9.12　独立供电电路图

2．轨迹检测模块

电动车前后各安装 3 只红外对管，一只置于轨道中间，两只置于轨道外侧。车体按轨迹行驶时，中间的红外传感器应检测到黑线（轨迹），两侧在轨迹外，不应检测到黑线，小车直走。但当小车脱离轨道时，即中间的红外传感器脱检测不到黑线，等待两侧的传感器中任一只检测黑线。当右侧的传感器检测到黑线时，说明车体偏左，应向右转；相反地，当左侧的传感器检测到黑线时，说明车体偏右，应向左转。这样，根据三只红外对管的检测状态，对车的方向做出相应的调整。因此，车体在行驶过程中，会有一定左右摇摆。当前面三对红外管均检测到轨迹时，说明车体已达到 B 端，可以暂停行驶。同样，当后面三对红外管均检测到轨迹时，说明车体已达到 A 端，也要停止行驶。

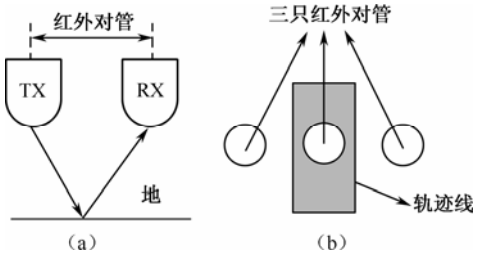


图 9.13　反射式红外传感器结构示意图

检测引导线寻迹传感器是一对反射式红外传感器，传感器检测原理如图 9.13（a）所示，TX 为发光二极管，RX 为光敏元件。为了能准确检测到轨迹线，按照如图 9.13（b）所示的方式分布。当传感器位于轨迹线上（黑色）时，对应红外对管的发光二极管灭，光敏三极管截止，检测电路输出低电平，反之则输出高电平。

轨迹检测电路部分原理图如图 9.14 所示。

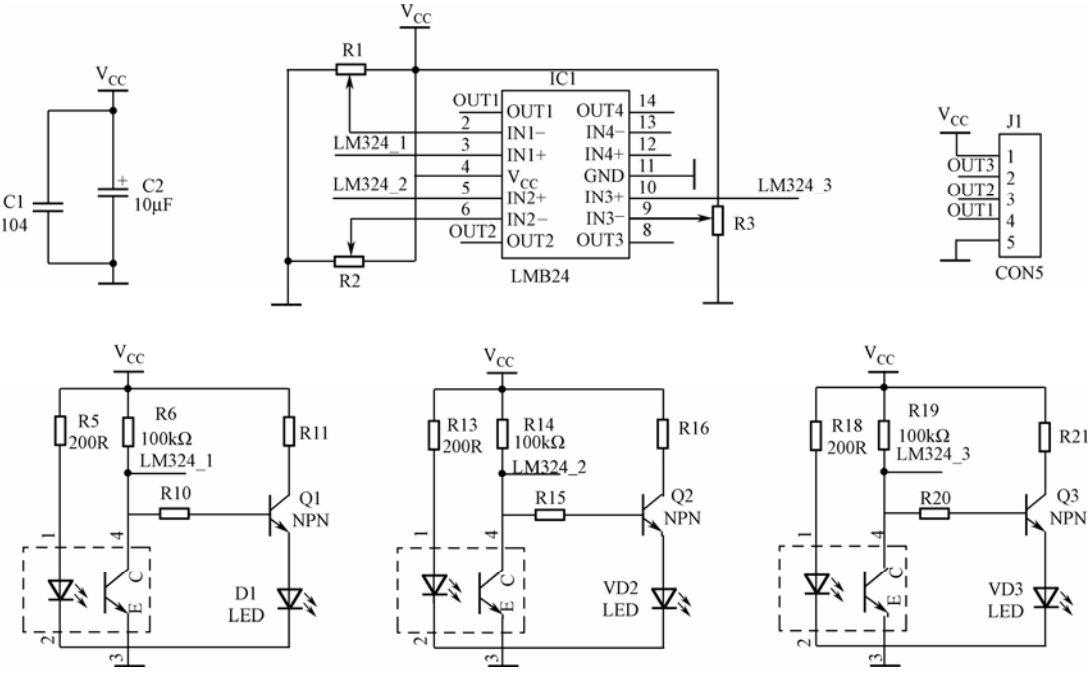


图 9.14　轨迹检测原理图

3．倾角检测模块

本系统选用 SCA61T 单轴倾角传感器。SCA61T 是芬兰 VTI 公司推出的一种高分辨率、低噪声、稳定性好、抗冲击能力强的单轴倾角传感器芯片。它能够测量−90° ～90° 范围的水平角度，具有 SPI 或模拟输出接口、5V 供电，比例电压输出等特点。芯片处于水平位置时，输出 2.5V，在倾斜时，以 0.03V/度的变化速率输出对应的电压值。使用该角度传感器完全可以满足系统对角度平衡控制的要求，而且硬件接口简单，编程也方便。

SCA61T 引脚图如图 9.15，引脚说明如表 9-1。其中，没有使用 SPI 接口时，SCK（引脚 1），MISO（引脚 2），MISI（引脚 3）和 CSB（引脚 5）都需要悬空。当 ST（引脚 6）为逻辑 1（供电电压）时，可激活自检，不用 ST 功能，引脚 9 悬空或者接地即可。选用比例电压输出方式来进行角度采集，不需 ST 功能，第 7 脚接入系统的模拟输入端即可。

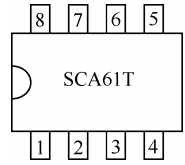


图 9.15　SCA61T 引脚图

4．电机驱动电路

采用专用芯片 L298N 作为电机驱动芯片。L298N 是一个具有高电压大电流的全桥驱动芯片，响应频率高，一片 L298N 可以分别控制两个电机，而且还带有控制使能端。用该芯片作为电机驱动，操作方便，稳定性好，性能优良，而且电路简单。L298N 的 ENA、ENB 端使能，5、7、10、12 引脚接到单片机 I/O 线上，通过对单片机的编程可以实现两个直流电机的正反转。具体电路图如图 9.16 所示。

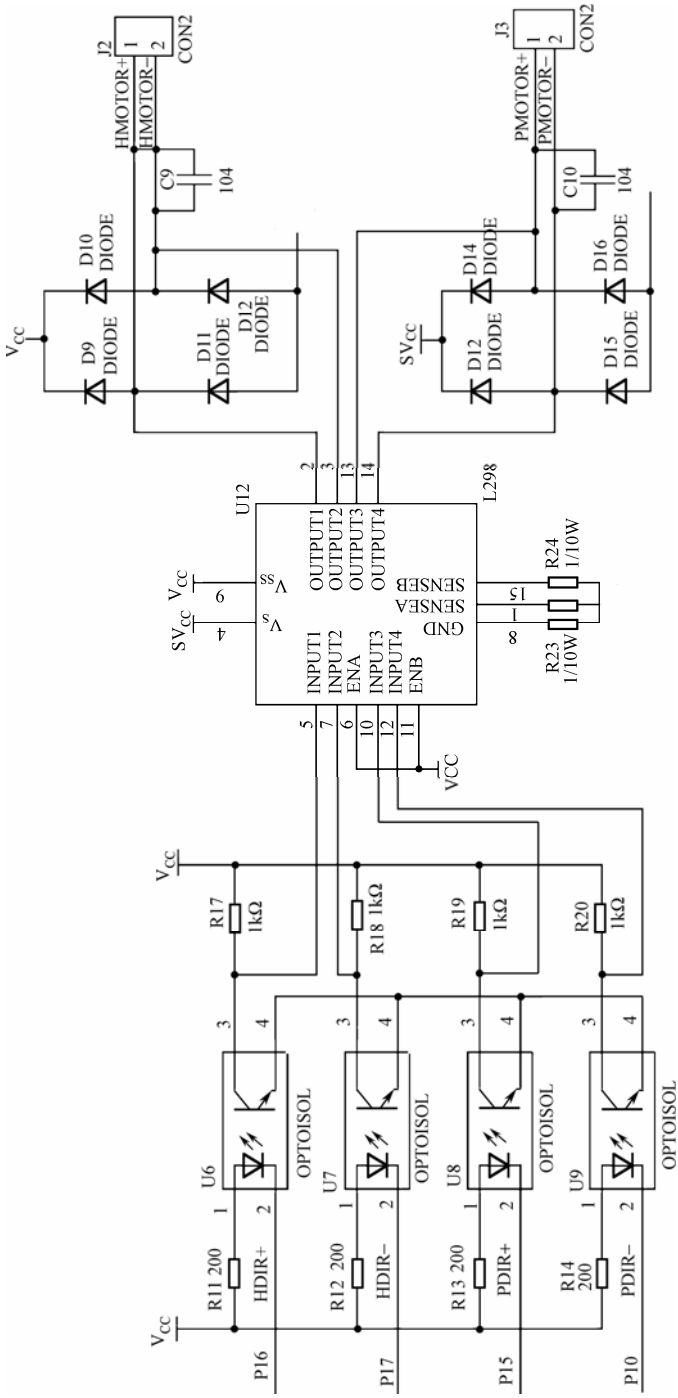


图 9.16　电机驱动电路图

表 9.1　SCA61T 引脚说明

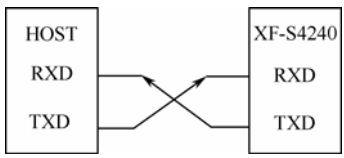
引　脚	名　称	I/O	连　接
1	SCK	输入	串行时钟
2	MISO	输出	主输入从输出，数据输出
3	MOSI	输入	主输出从输入，数据输入
4	GND	电源	电源地
6	ST	输入	自检输入
7	OUT	输出	模拟输出
8	VDD	电源	电源供电

5．显示模块

采用台湾矽创电子公司生产的 ST7920 驱动液晶显示器，ST7920 芯片内置 8192 个中文汉字（16×16 点阵）、128 个字符的 ASCII 字符库（8×16 点阵）。128×64 液晶显示清晰、快速、信息量大、使用方便。ST7920 驱动 LCD 本身带有简单中文字库，可显示 4 行 8 列汉字，汉字显示可直接提示信息含义，具有人性化特点。ST7920 可提供 4 位并行、8 位并行、2 线串行以及 3 线串行等多种接口方式，在本系统中采用 3 线串行接口方式，接口电路简单，编程方便。

6．声光报警模块

声音提示使用安徽中科大讯飞信息科技有限公司研发的 XF-S4240 语音合成模块。XF-S4240 模块采用 COB（Chips On Board）封装，可方便地集成到需要中文语音合成功能的嵌入式设备之中。XF-S4240 可通过 UART、SPI 和 I^2C 三种接口接收待合成的文本，直接合成为语音输出。本系统采用异步串口 UART 接受单片机控制系统的命令和数据，允许发送数据的最大长度为 1K 字节。XF-S4240 连接示意图如图 9.17 所示。



错误！

7．单片机系统

综合以上各模块对单片机接口的功能要求，选用单片机 STC5412AD 作为主控制芯片。STC5412AD 是一种兼容的增强型 51 微控制器，是高速/低功耗的新一代 8051 单片机，内部集成的 8 通道 10 位 ADC，能够满足本系统的需要 A/D 转换等要求。单片机控制系统原理图如图 9.18 所示。

9.5.2　软件设计

小车平衡的理想目标是跷跷板的水平角度为 0°，但要达到这一目标是十分困难的，只要跷跷板的 A、B 两端的绝对高度差小于 40mm 就认为达到平衡状态。跷跷板平衡是一个过程，需要一定时间，所以不能只根据跷跷板的瞬时水平角度来作为小车前进或后退的依据，还要判断跷跷板水平角度变化率。

软件系统主要包括人机界面、寻迹、倾角检测、电机控制、平衡控制、语音播报等部分。为完成以上功能，使用了单片机 A/D、TIMER，USART、I/O、EEPROM 等资源，其中串口实现语音播报功能、A/D 实现倾角采集功能、定时器实现各阶段行驶时间的计时功能、I/O 实现液晶显示、电机控制、按键等功能、EEPROM 实现调试参数保存功能。

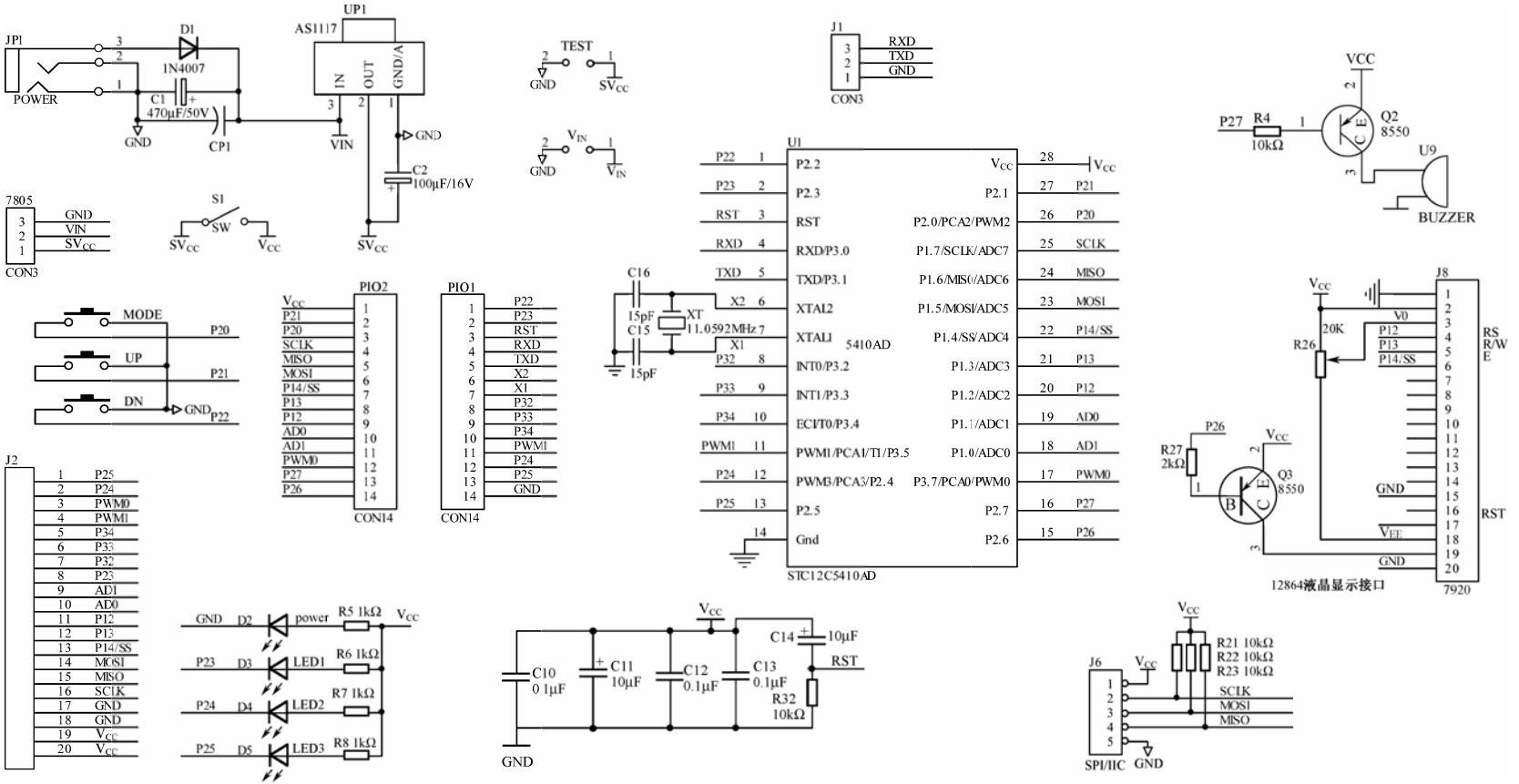


图 9.18　单片机控制系统原理图

系统设计了五种界面：0 页面显示基本任务的各阶段时间；1 页面显示实时角度及行驶总时间；2 页面显示水平角度值，用来保存跷跷板平衡时的角度（由于传感器安装时存在水平位置的偏差，因此跷跷板平衡时，传感器不见得是水平 0 角度）；3 页面调整基本任务时从 A 点到达 C 点左右的大致时间，4 页面调整发挥任务时从地面到达平衡点左右的大致时间。　三个按键用来切换界面并保存显示相关参数。MODE 键用来切换界面，UP 键用来调整参数或启动行驶，DOWN 键用来调整参数。按键检测界面显示可用定时器实时扫描以更新相关信息。定时器程序流程图如图 9.19 所示。

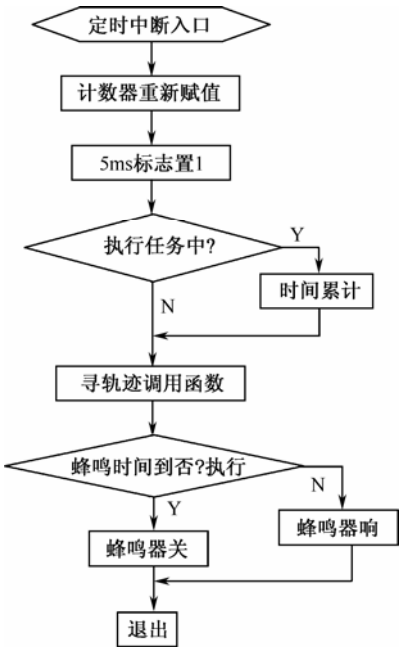


图 9.19　定时器中断流程图

EEPROM 用来存储平衡角度、行驶时间等相关参数，这样防止每次上电，参数丢失。

车体在行驶过程中，需不断判断是否偏离轨迹线，因此，寻迹和电机控制可以在定时器中完成，即每 5ms 检测一次相对位置，并进行方向调整。小车寻迹程序流程图如图 9.20 所示。

基于平衡控制思想，在车体未到达平衡点左右时，无需检测角度，只有在平衡附近时才进行检测判断。本系统要实现的任务分为基本任务和发挥任务，其程序流程图如图 9.21 和图 9.22 所示。

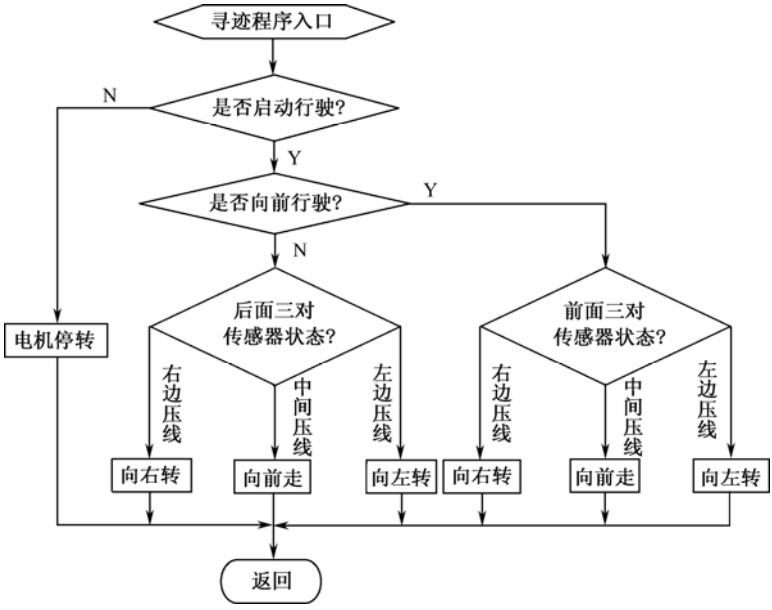


图 9.20　小车寻迹流程图

另外，车体在任务执行过程中，不接收其他操作，如按键切换界面，参数设置等。主程序流程图如图 9.23 所示。

软件系统包括 6 个工程文件，为了完整，详细列出各工程文件清单。

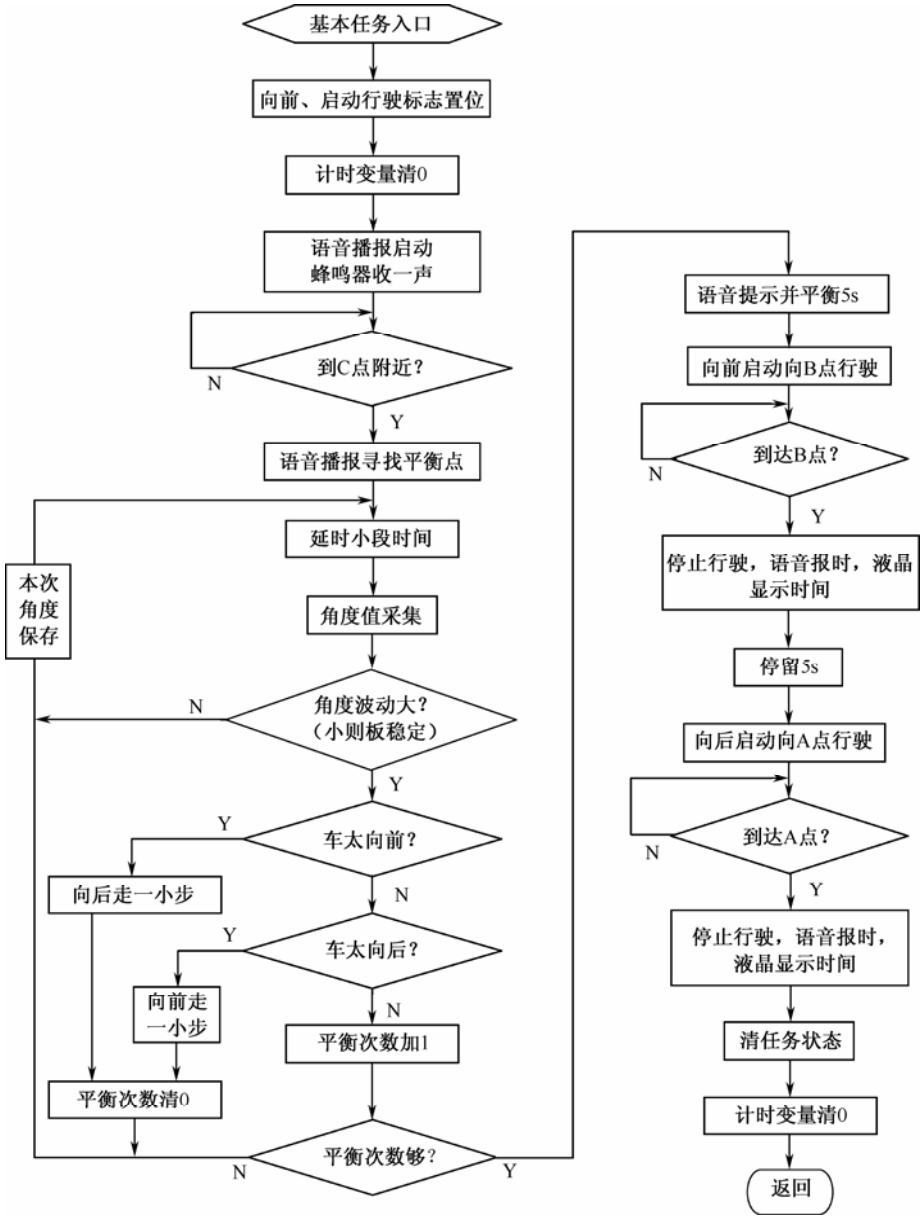


图 9.21　基本任务执行流程图

工程文件 1 为“HEAD.H”，主要定义系统用到的 I/O 引线和标志位，其源代码如下：

```
sbit Mode = P2^0;          //MODE 按键 IO 定义
sbit Up   = P2^1;          //UP 按键 IO 定义
sbit Down = P2^2;          //DOWN 按键 IO 定义
sbit LZMove = P1^6;        //前左电机控制 IO 定义
sbit LFMove = P1^7;        //后左电机控制 IO 定义
sbit RZMove = P1^5;        //前右电机控制 IO 定义
sbit RFMove = P1^0;        //后右电机控制 IO 定义
sbit Led_left = P3^3;      //前方左边寻迹传感器
sbit Led_qian = P3^4;      //前方中间寻迹传感器
```

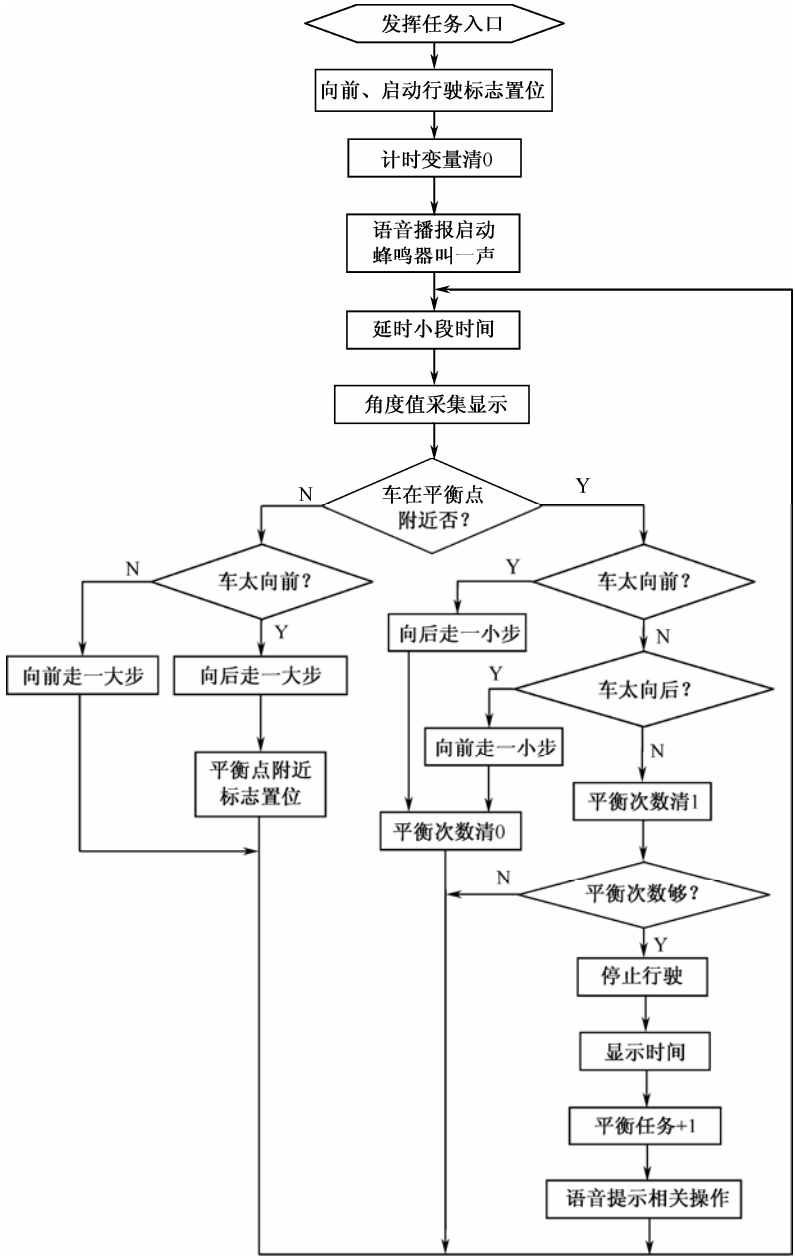


图 9.22　发挥任务执行流程图

```
sbit Led_right = P2^3;        //前方右边寻迹传感器
sbit Led_leftH = P2^4;        //后方左边寻迹传感器
sbit Led_qianH = P2^5;        //后方中间寻迹传感器
sbit Led_rightH= P3^7;        //后方右边寻迹传感器
sbit Sound = P2^7;            //蜂鸣器 IO 定义
#define Renwu0_ing 1
#define Renwu0     2
#define Renwu1_ing 3
```

```
#define Renwu1      4
#define    Black 1              //红外传器压黑线时的 IO 有效位定义
#define    White0
#define Froth 0                 //车体行驶方向定义
#define Back    1
#define Move 1                  //电机转动有效位定义
#define Stop 0
```

工程文件 2 为“LCD.C”，完成液晶显示，其源代码如下：

```
#include "reg5410.h"
sbit CS =P1^2;
sbit SID=P1^3;
sbit SCLK=P1^4;
void delay(unsigned int j)
{
    unsigned char i;
    do{
    for(i=0; i<100; i++);
    }while(j--);
}
void send_command(unsigned char command_data)
{
    unsigned char i;
    unsigned char i_data,temp_data1,temp_data2;
    i_data=0xf8;
    delay(10);
    CS=1;
    SCLK=0;
    for(i=0; i<8; i++)
    {
        SID=(bit)(i_data&0x80);
        SCLK=0;
        SCLK=1;
        i_data=i_data<<1;
    }
    i_data=command_data;
    i_data&=0xf0;
    for(i=0; i<8; i++)
    {
        SID=(bit)(i_data&0x80);
        SCLK=0;
        SCLK=1;
```

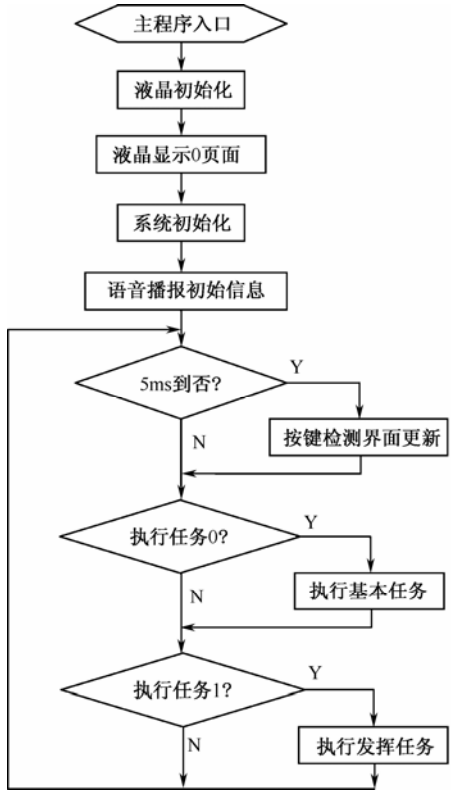


图 9.23　主程序流程图

```
            i_data=i_data<<1；
        }
        i_data=command_data；
        temp_data1=i_data&0xf0；
        temp_data2=i_data&0x0f；
        temp_data1>>=4；
        temp_data2<<=4；
        i_data=temp_data1|temp_data2；
        i_data&=0xf0；
        for(i=0；i<8；i++)
        {
            SID=(bit)(i_data&0x80)；
            SCLK=0；
            SCLK=1；
            i_data=i_data<<1；
        }
        CS=0；
}
void send_data(unsigned char command_data)
{
        unsigned char i；
        unsigned char i_data,temp_data1,temp_data2；
        i_data=0xfa；
        delay(10)；
        CS=1；
        for(i=0；i<8；i++)
        {
            SID=(bit)(i_data&0x80)；
            SCLK=0；
            SCLK=1；
            i_data=i_data<<1；
        }
        i_data=command_data；
        i_data&=0xf0；
        for(i=0；i<8；i++)
        {
            SID=(bit)(i_data&0x80)；
            SCLK=0；
            SCLK=1；
            i_data=i_data<<1；
        }
```

```
    i_data=command_data；
    temp_data1=i_data&0xf0；
    temp_data2=i_data&0x0f；
    temp_data1>>=4；
    temp_data2<<=4；
    i_data=temp_data1|temp_data2；
    i_data&=0xf0；
    for(i=0；i<8；i++)
    {
        SID=(bit)(i_data&0x80)；
        SCLK=0；
        SCLK=1；
        i_data=i_data<<1；
    }
    CS=0；
}
void InitLCD()                    //液晶初始化
{
    send_command(0x30)；    //功能设置：一次送 8 位数据，基本指令集
    send_command(0x06)；    //点设定：显示字符/光标从左到右移位，DDRAM 地址加 1
    send_command(0x0c)；    //显示设定：开显示，显示光标,当前显示位反白闪动
    send_command(0x04)；    //显示设定：开显示，显示光标，当前显示位反白闪动
    send_command(0x01)；    //清 DDRAM
    send_command(0x02)；    //DDRAM 地址归位
    send_command(0x80)；    //把显示地址设为 0X80，即为第一行的首位
}
//汉字显示，x，y 为起始坐标 x(0<=x<=3)，y(0<=y<=7)，x 为行坐标，y 为列坐标；how 为要显示汉字的个数；str 是要显示汉字的地址
void DispHanzi(unsigned char x,unsigned char y,unsigned char how,unsigned char *stri)
{
    unsigned char hi=0；
    if(x==0) send_command(0x80+y)；
    else if(x==1) send_command(0x90+y)；
    else if(x==2) send_command(0x88+y)；
    else if(x==3) send_command(0x98+y)；
    for(hi=0；hi<how；hi++)
    {
        send_data(*(stri+hi*2))；
        send_data(*(stri+hi*2+1))；
    }
}
```

```
//字母显示
void DispZimu(unsigned char x,unsigned char y,unsigned char how,unsigned char *stri)
{
    unsigned char hi=0；
    if(x==0) send_command(0x80+y)；
    else if(x==1) send_command(0x90+y)；
    else if(x==2) send_command(0x88+y)；
    else if(x==3) send_command(0x98+y)；
    for(hi=0；hi<how；hi++)
    {
        send_data(0xA3)；
        send_data(*(stri+hi))；
    }
}
//8*16 数字显示： x，y 为起始坐标 x(0<=x<=3)，y(0<=y<=7)，x 为行坐标，y 为列坐标；n 为显示数字值
void DispShuzi(unsigned char x,unsigned char y,unsigned int n)
{
    if(x==0) send_command(0x80+y)；
    else if(x==1) send_command(0x90+y)；
    else if(x==2) send_command(0x88+y)；
    else if(x==3) send_command(0x98+y)；
    if(n/1000%10!=0)
    {
        send_data(0x30+n/1000%10)；
        send_data(0x30+n/100%10)；
        send_data(0x30+n/10%10)；
        send_data(0x2E)；
        send_data(0x30+n%10)；
        send_data(0x20)；
        send_data(0x73)；
    }
    else if(n/100%10!=0)
    {
        send_data(0x20)；
        send_data(0x30+n/100%10)；
        send_data(0x30+n/10%10)；
        send_data(0x2E)；
        send_data(0x30+n%10)；
        send_data(0x20)；
        send_data(0x73)；
```

```
        }
        else if(n/10%10!=0)
        {
            send_data(0x20);
            send_data(0x20);
            send_data(0x30+n/10%10);
            send_data(0x2E);
            send_data(0x30+n%10);
            send_data(0x20);
            send_data(0x73);
        }
        else
        {
            send_data(0x20);
            send_data(0x20);
            send_data(0x30);
            send_data(0x2E);
            send_data(0x30+n%10);
            send_data(0x20);
            send_data(0x73);
        }
    }
```

工程文件 3 为“EEPROM.C”，完成串行 EEPROM 的信息读写，其源代码代码如下：

```
    #include "reg5410.h"
    unsigned char read(unsigned char ADDRH,unsigned char ADDRL);
    void write(unsigned char ADDRH,unsigned char *p,unsigned char k);
    void erase(unsigned char ADDRH);
    void Delay0(unsigned int n)
    {
        unsigned char i,j;
        for(i=0;i<n;i++)
            for(j=0;j<100;j++);
    }
    //STC 芯片内部 EEPROM 写函数
    void write(unsigned char ADDRH,unsigned char *p,unsigned char k)
    {
        unsigned char i;
        erase(ADDRH);
        ISP_ADDRH=ADDRH;                //擦除
        ISP_ADDRL=0x00;
        for(i=0;i<k;i++)
```

```
        {
            ISP_DATA=p[i];
            ISP_CONTR=0x8b;
            ISP_CMD=2;
            ISP_TRIG=0x46;                  //触发 ISP/IAP
            ISP_TRIG=0xb9;
            ISP_ADDRL++;                    //地址+1;
            Delay0(5);
        }
    }
    //STC 芯片内部 EEPROM 读函数：ADDRH  范围 0x28-0x2f   ADDL“0～255”
    unsigned char read(unsigned char ADDRH,unsigned char ADDRL)
    {
        ISP_ADDRH=ADDRH;                    //ISP/IAP 控制寄存器
        ISP_ADDRL=ADDRL;
        ISP_CONTR=0x8b;
        ISP_CMD=1;                          //送字节读命令
        ISP_TRIG=0x46;
        ISP_TRIG=0xb9;                      //触发 ISP/IAP
        Delay0(5);
        return ISP_DATA;
    }
    //STC 芯片内部 EEPROM 擦除函数
    void erase(unsigned char ADDRH)
    {
        ISP_ADDRH=ADDRH;
        ISP_ADDRL=0x00;
        ISP_CONTR=0x8b;
        ISP_CMD=3;
        ISP_TRIG=0x46;                      //触发 ISP/IAP
        ISP_TRIG=0xb9;
        Delay0(5);
    }
```

工程文件 4 为“KEYDEL.C”，完成按键处理，其源代码代码如下：

```
    #include "reg5410.h"
    #include "head.h"
    extern void DispZimu(unsigned char x,unsigned char y,unsigned char how,unsigned char *stri);
    extern void DispShuzi(unsigned char x,unsigned char y,unsigned int n); //8*16 数字显示
    extern void DispHanzi(unsigned char x,unsigned char y,unsigned char how,unsigned char *stri);
    extern void send_command(unsigned char command_data) ;
    extern void GetADCaiJi(unsigned int *ADData);
```

```
extern void write(unsigned char ADDRH,unsigned char *p,unsigned char k);
extern unsigned char work;
extern unsigned int Maxjiao;
extern unsigned int Minjiao;
extern unsigned int j_time;
extern unsigned int f_time;
void Disp_Page0(void);                    //显示 0 页面函数定义
void Disp_Page1(void);                    //显示 1 页面函数定义
void Disp_Page2(void);                    //显示 2 页面函数定义
void Disp_Page3(void);                    //显示 3 页面函数定义
void Disp_Page4(void);                    //显示 4 页面函数定义
void Disp_Page5(void);                    //显示 5 页面函数定义
void jiemian(void)
{
    static unsigned char ti=0,page=0;
    unsigned int tmp=0;
    unsigned char dat[2];
    if(page==2)                           //2 页面需要显示角度值
        {
            GetADCaiJi(&tmp);
            DispShuzi(1,0,tmp);
        }
    if(Mode==0||Up==0||Down==0)           //有键按下
    {
        ti++;
        if(ti>=30)                        //消抖
        {
            ti=0;
            if(Mode==0)                   //MODE 键切换界面
            {
                page++;
                if(page>=5)
                    page=0;
                if(page==0)
                    Disp_Page0();
                else if(page==1)
                    Disp_Page1();
                else if(page==2)
                    Disp_Page2();
                else if(page==3)
                    Disp_Page4();
```

```
                else if(page==4)
                    Disp_Page5();
            }
            else if(Up==0)              //UP 按键处理
            {
            if(page==0)
                {work=Renwu0_ing；Disp_Page0()；} //0 页面时，按下 UP 键，启动基本任务操作
                else if(page==1)
                {work=Renwu1_ing；Disp_Page1()；} //1 页面时，按下 UP 键，启动发挥任务操作
                else if(page==2)        //2 页面，按下 UP 键，保存平衡时小车的角度
                {
Maxjiao=tmp；dat[0]=Maxjiao/256；dat[1]=Maxjiao%256； write(0x28,dat,2)；DispShuzi(3,0,Maxjiao)；
}
                else if(page==3)        //基本任务，到达 C 点的大致时间修改
                {
j_time=j_time+20；if(j_time>1800)j_time=0；DispShuzi(1,2,j_time)；dat[0]=j_time/256；dat[1]=j_time%256；
write(0x2C,dat,2)；
}
                else if(page==4)        //发挥任务，到达平衡点附近的大致时间修改
                {
f_time=f_time+20； if(f_time>1800)f_time=0； DispShuzi(1,2,f_time)； dat[0]=f_time/256；
dat[1]=f_time%256；
write(0x2E,dat,2)；
}
            }
            else if(Down==0)            //Down 按键处理
            {
                if(page==2)             //2 页面，保存平衡时小车的角度
                {
Maxjiao=tmp； dat[0]=Maxjiao/256； dat[1]=Maxjiao%256； write(0x28,dat,2)；
}
                else if(page==3)        //基本任务，到达 C 点的大致时间修改
                {
if(j_time>=20) j_time=j_time-20；
if(j_time<20)j_time=1800； DispShuzi(1,2,j_time)； dat[0]=j_time/256； dat[1]=j_time%256；
write(0x2C,dat,2)；
}
                else if(page==4)        //发挥任务，到达平衡点附近的大致时间修改
                {
if(j_time>=20) f_time=f_time-20；
if(f_time<20)f_time=1800； DispShuzi(1,2,f_time)； dat[0]=f_time/256； dat[1]=f_time%256；
write(0x2E,dat,2)；
```

```
}
                    }
                }
            }
            else ti=0;
}
void Disp_Page0(void)
{
            send_command(0x01);
            DispZimu(0,0,4,"A->C");
            DispZimu(1,0,4,"C->C");
            DispZimu(2,0,4,"C->B");
            DispZimu(3,0,4,"B->A");
}
void Disp_Page1(void)
{
            send_command(0x01);
            DispHanzi(0,0,4,"实时角度");
            DispHanzi(2,0,3,"总时间");
}
void Disp_Page2(void)
{
            send_command(0x01);
            DispHanzi(0,1,5,"水平角度值");
}
void Disp_Page4(void)
{
            send_command(0x01);
            DispHanzi(0,1,5,"基本总时间" );
            DispShuzi(1,2,j_time);
}
void Disp_Page5(void)
{
            send_command(0x01);
            DispHanzi(0,1,5,"发挥总时间");
            DispShuzi(1,2,f_time);
}
```

工程文件 5 为“RENWU.C”，完成数据采集和小车运动控制，分为基本任务、发挥任务和语音播报等，其源代码代码如下：

```
#include "reg5410.h"
#include "head.h"
```

```
unsigned int Maxjiao=502；
unsigned int Minjiao=483；
unsigned int j_time=940；
unsigned int f_time=1500；
extern void DispShuzi(unsigned char x,unsigned char y,unsigned int n)；      //8*16 数字显示
extern void send_data(unsigned char command_data) ；
void yuyin(unsigned char n,unsigned char *str)；
void yuyin1(unsigned char n,unsigned char *str,unsigned int x,unsigned char n1,unsigned char *st)；
extern unsigned char five_ms；
extern unsigned char Sound_time；
extern unsigned int Run_time；
extern unsigned int delay；
extern unsigned char work；
bit Dir=0；                          //行驶方向标志定义，Back：向后，Front：向前
bit Start=0；                        //车体行驶标志，1：行驶，0：停止
void Find_Way(void)
{
    if(Start==1)                     //车体需要前行
    {
        if(Dir==Froth)               //目前是向前行驶
        {
            if(Led_left==Black)    //左边是否压线，压线向左转
            {    LZMove=1；RZMove=0；    }
            else if(Led_right==Black)          //右边是否压线，压线向右转
            {    LZMove=0；RZMove=1；    }
            else if(Led_qian==Black)           //中间是否压线，压线方向不变
            {    LZMove=0；RZMove=0；    }
        }
        else if(Dir==Back)                     //目前是向后行驶
        {
            if(Led_leftH==Black)               //左边是否压线，压线向左转
            {    LFMove =1；RFMove=0；}
            else if(Led_rightH==Black)         //右边是否压线，压线向右转
            {    LFMove =0；RFMove=1；}
            else if(Led_qianH==Black)          //中间是否压线，压线方向不变
            {    LFMove =0；RFMove=0；}
        }
    }
    else
    {    LZMove=1；      RZMove=1；      LFMove =1；      RFMove=1；}
}
```

```
//A/D 采集函数，指针变量返回采集代码值 0-1024
void GetADCaiJi(unsigned int *ADData)
{
    unsigned int i, j, Sum；
    ADC_CONTR=0x80|0x01；
    for(i=0；  i<100；  i++)；
    Sum = 0；
    for(i=0；  i<64；  i++)
    {
        ADC_CONTR|=0x08；
        while((ADC_CONTR&0x10)==0x00)； //等待 A/D 转换结束
        Sum = Sum + (ADC_DATA*4+ADC_LOW2)；
        for(j=0；  j<100；j++)；
    }
    *ADData = Sum / 64；                    //滤波处理
}
//基本任务操作
void Do_Renwu0(void)
{
    unsigned int tmp=0,lasttmp=0,tim；
    unsigned char Pingheng=0；
    unsigned char dei=10；                  //运动时间
    unsigned int delayi=200,j=0,ji=0；
    unsigned char PH=0；
    Run_time=0；
    Dir=Froth；  Start=1；                  //向前，启动行驶标志置位
    LZMove=0；RZMove=0；                    //向前走
    Run_time=0；                            //重新开始计时
    yuyin(15,"开始!!向 C 点行驶")；          //语音播放
    Sound_time=50；                         //蜂鸣器叫一声，表示开始
    while(Run_time<j_time)；                //向前行驶时间到，该时间需经调试来确定
    tim=Run_time/20；
    LZMove=1；      RZMove=1；      LFMove =1；      RFMove=1；//停止行驶
    DispShuzi(0,4,tim)；                    //显示时间
    Sound_time=20；                         //蜂鸣器叫一声
    yuyin1(12,"到达 C 点|用时"，tim,10，"寻找平衡点")；      //语音提示
    Pingheng=0；                            //平衡标志清 0
    Run_time=0；                            //平衡时间清 0
    while(1)                                //寻找平衡过程
    {
        Start=0；
```

```
            delay=50；
            while(delay)；
            GetADCaiJi(&tmp)；                    //角度采集
            if(tmp<lasttmp+7||tmp>lasttmp-7)
//与上次采集偏差较小，认为板较稳定，需进行前后调整
            {
                if(tmp>(Maxjiao+6))               //太向后了，向前走一小步
                {   Dir = Froth ；  Start = 1 ；  //启动向前走
                    delay=3；                     //行驶时间定义
                    LZMove=0；RZMove=0；          //开始行驶
                    while(delay)；                //行驶时间到
                    Start=0；                     //停止行驶
                    LZMove=1；      RZMove=1；LFMove =1；     RFMove=1；
                    Pingheng=0；                  //平稳标志为 0
                }
                else if(tmp<(Maxjiao-6))          //太向前了，向后走一小步
                {
                    Dir = Back；  Start = 1；     //启动向后走
                    delay=3；
                    LFMove=0；RFMove=0；          //开始行驶
                    while(delay)；
                    Start=0；                     //行驶时间到，停止
                    LZMove=1；      RZMove=1；LFMove =1；     RFMove=1；
                    Pingheng=0；
                }
                else                              //在平衡角度范围内
                {   Pingheng++；                  //平衡时间计算
                    if(Pingheng>=20)
//平衡次数足够，认为平衡，停止行驶，显示时间，并退出寻找平衡过程
{Start=0；LZMove=1；RZMove=1；LFMove=1；RFMove=1； tim=Run_time/20；DispShuzi(1,4,tim)；
break；}
                }
            }
            lasttmp=tmp；
        }
        yuyin1(15,"到达平衡点|用时",tim,0,"")；     //语音提示平衡
        delay=1000；
        while(delay)                              //平衡时间为 5s
        {
            if(delay%200==190)
                Sound_time=10；
```

```
        }
        yuyin(11,"向|B 点|行驶");                    //向 B 点行驶
        Run_time=0;
        Dir=Froth;  Start=1;
        LZMove=0; RZMove=0;
        while(Led_left==White||Led_right==White||Led_qian==White);        //等待到达 B 点
        Start=0;
        LZMove=1;      RZMove=1;      LFMove =1;      RFMove=1;      //到达 B 点,停止行驶
        tim=Run_time/20;
        DispShuzi(2,4,tim);
        yuyin1(13,"到达|B 点|用时",tim,0," ");       //报时
        delay =1000;
        while(delay)                                 //在 B 点停留 5s
        {
            if(delay%200==190)
                Sound_time=10;
        }
        yuyin(10,"向|A 点行驶");                     //向 A 点行驶
        Run_time=0;
        Dir = Back; Start = 1;
        LFMove=0; RFMove=0;
        while(Led_leftH==White||Led_rightH==White||Led_qianH==White);    //等待到达 A 点
        Start = 0;                                   //到达 A 点，声音报时，并指示任务完成等
        Sound_time=50;
        tim=Run_time/20;
        DispShuzi(3,4,tim);
        yuyin1(12,"到达 A 点|用时",tim,18,"|基本部奋|测试完毕");
        work=Renwu0;
        Run_time=0;
        LZMove=1;      RZMove=1;      LFMove =1;      RFMove=1;
}
//发挥任务操作
void Do_Renwu1(void)
{
        unsigned char flagph=0,Pingheng=0,i=0,PH=0;
        unsigned int tmp=0,lasttmp=0,jiaodu=0,tim=0;
        unsigned char si=0,phi=0;                    //phi 为平衡次数
        unsigned char dei=20;                        //运动时间
        unsigned int delayi=20,j=0,ji=0;
        yuyin(18,"发挥部奋|测试|开始");
        for(j=0; j<520; j++)
```

```
    for(ji=0；ji<5000；ji++)；              //延时
Run_time = 0；
Dir = Froth ；  Start = 1；                 //向前行驶
delayi=200；
dei=20；
Sound_time=50；
LZMove=0；RZMove=0；
yuyin(12,"向|跷板|行驶")；
while(Run_time<f_time)；                    //一次到达平衡点左右的时间到否
LZMove=1；      RZMove=1；      LFMove =1；     RFMove=1；      //到了，停止行驶
while(1)                                    //寻找平衡
{
    Start=0；
    LZMove=1；      RZMove=1；LFMove =1；     RFMove=1；
    if(tmp>Maxjiao)
    {
        jiaodu=1.25*(tmp-Maxjiao)；
        DispShuzi(1,1,jiaodu)；
    }
    else if(tmp<=Maxjiao)
    {
        jiaodu=1.25*(Maxjiao-tmp)；
        DispShuzi(1,1,jiaodu)；             //显示平衡角度
    }
    if(PH==0)                               //未在平衡点附近
    {
        delay=50；
        while(delay)；                      //100ms
        GetADCaiJi(&tmp)；
        if(tmp>(Maxjiao-20))                //角度离平衡角度相关较远,
        {
            Dir=Froth；Start=1；            //大步向前走
            LZMove=0；RZMove=0；
            delay=20；
            while(delay)；
            Start=0；
            LZMove=1；      RZMove=1；      LFMove =1；     RFMove=1；
        }
        else                                //角度离平衡角度接近
        {
            Dir=Back；Start=1；             //向后退 1 秒
```

```
            LFMove=0；RFMove=0；
            delay=30；
            while(delay)；
            Start=0；
            LZMove=1；      RZMove=1；      LFMove =1；      RFMove=1；
            PH=1；                  //处于平衡点附近
        }
        }
    else if(PH==1)                  //平衡点附近，进行细调
    {
        delay=50；
        while(delay)；              //100ms
        GetADCaiJi(&tmp)；
        if( (tmp<lasttmp+4)||(tmp>lasttmp-4))       //角度有一定偏差，小步调整
        {
            if(tmp>(Maxjiao+6))     //车太向后了，需向前走
            {
                Dir=Froth；Start=1；
                LZMove=0；RZMove=0；
                delay=3；
                while(delay)；
                Start=0；
                LZMove=1；RZMove=1；LFMove =1；RFMove=1；
                Pingheng=0；
                if(flagph==1)
                PH=0；
                flagph=0；
            }
            else if(tmp<(Maxjiao-6))
            {
                Dir=Back；Start=1；     //车太向前了，需向后走
                delay=3；
                LFMove=0；RFMove=0；
                while(delay) ；
                Start=0；
                LZMove=1；      RZMove=1；      LFMove =1；      RFMove=1；
                Pingheng=0；
                if(flagph==1)
                    PH=0；
                flagph=0；
            }
```

```
                    else                        //不前不后，则处于平衡
                    {   Pingheng++;
                        if(Pingheng>=25)        //平衡次数足够，认为找到平衡点
                        {
                            Start=0;            //停止行驶
                            LZMove=1;    RZMove=1;    LFMove =1;    RFMove=1;
                            if(flagph==0)
                            {
                                tim=Run_time/20;
                                DispShuzi(3,1,tim);
                                if(phi==0)
                                    yuyin1(15,"第一次平衡|用时",tim,9,"请放|重物");
                                else if(phi==1)
                                    yuyin1(15,"第二次平衡|用时",tim,0,"");
                                phi++;
                                flagph=1;
                                delay = 1600;
                                while(delay)
                                {
                                    if(delay%200==0)
                                    Sound_time=10;
                                }
                            }
                        }
                    }
                }
                lasttmp=tmp;
            }
        }
}
//语音播报函数，N 为汉字内码字节数，str 为汉字内码值
void yuyin(unsigned char n,unsigned char *str)
{
        unsigned char i=0;
        SBUF=0xFE;
        while(TI==0);
        TI=0;
        SBUF=0x01;
        while(TI==0);
        TI=0;
        SBUF=0x00;
        while(TI==0);
```

```
        TI=0;
        SBUF=n;
        while(TI==0);
        TI=0;
        for(i=0; i<n; i++)
        {
            SBUF=*(str+i);
            while(TI==0);
            TI=0;
        }
}
void yuyin1(unsigned char n,unsigned char *str,unsigned int x,unsigned char n1,unsigned char *st)
{
        unsigned char i=0,sen[4],xi=0;
        if(x>999)
        {
            sen[0]=x/1000%10;
            sen[1]=x/100%10;
            sen[2]=x/10%10;
            sen[3]=x%10;
            xi=4;
        }
        else if(x>99)
        {
            sen[0]=x/100%10;
            sen[1]=x/10%10;
            sen[2]=x%10;
            xi=3;
        }
        else if(x>9)
        {
            sen[0]=x/10%10;
            sen[1]=x%10;
            xi=2;
        }
        else
        {
            sen[0]=0;
            sen[1]=x%10;
            xi=2;
        }
        SBUF=0xFE;
```

```
while(TI==0);
TI=0;
SBUF=0x01;
while(TI==0);
TI=0;
SBUF=0x00;
while(TI==0);
TI=0;
SBUF=n+xi+12+n1;
while(TI==0);
TI=0;
for(i=0; i<n; i++)
{
    SBUF=*(str+i);
    while(TI==0);
    TI=0;
}
SBUF='[';
while(TI==0);
TI=0;
SBUF='n';
while(TI==0);
TI=0;
SBUF='2';
while(TI==0);
TI=0;
SBUF=']';
while(TI==0);
TI=0;
for(i=0; i<xi; i++)
{
    SBUF=sen[i]+48;
    while(TI==0);
    TI=0;
    if(xi==(i+2))
    {
        SBUF=0xb5;
        while(TI==0);
        TI=0;
        SBUF=0xe3;
        while(TI==0);
```

```
                TI=0;
                SBUF='[';
                while(TI==0);
                TI=0;
                SBUF='n';
                while(TI==0);
                TI=0;
                SBUF='2';
                while(TI==0);
                TI=0;
                SBUF=']';
                while(TI==0);
                TI=0;
            }
        }
        SBUF=0xc3;
        while(TI==0);
        TI=0;
        SBUF=0xeb;
        while(TI==0);
        TI=0;
        for(i=0; i<n1; i++)
        {
            SBUF=*(st+i);
            while(TI==0);
            TI=0;
        }
    }
```

工程文件 6 为“MAINFILE.C”，是主程序，完成系统初始化，其源代码如下：

```
#include "reg5410.h"
#include "head.h"
extern void InitLCD();
extern void jiemian(void);
extern void Disp_Page0(void);
extern void Do_Renwu0(void);
extern void Do_Renwu1(void);
extern unsigned char read(unsigned char ADDRH,unsigned char ADDRL);
extern void write(unsigned char ADDRH,unsigned char *p,unsigned char k);
extern void yuyin(unsigned char n,unsigned char *str);
extern unsigned int Maxjiao;
extern unsigned int Minjiao;
```

```
extern unsigned int j_time;
extern unsigned int f_time;
extern void Find_Way(void);
void System_init(void);
unsigned char five_ms=0;
unsigned char work=0;
unsigned char Sound_time=0;
unsigned int Run_time=0;
unsigned int delay=0;
main()
{
    InitLCD();                  //液晶初始化
    Disp_Page0();               //显示 0 页
    System_init();              //系统初始化
    yuyin(13,"电动车|跷跷板");   //上电语音播报"电动车跷跷板"
    while(1)
    {
        if(five_ms == 1) {five_ms=0; jiemian();  } //界面参数每 5ms 更新一次
        if(work == Renwu0_ing)   Do_Renwu0();   //是否执行任务 0，即基本功能，是则执行
        if(work ==    Renwu1_ing) Do_Renwu1(); //是否执行任务 1，即发挥部分功能，是则执行
    }
}
void System_init(void)
{
    unsigned int ti=0;
    ti=read(0x28,0x00);
    Maxjiao=ti*256+read(0x28,0x01);          //平衡时的最大角度
    ti=read(0x2A,0x00);
    Minjiao=ti*256+read(0x2A,0x01);          //平衡时的最小角度
    j_time=980;
    ti=read(0x2E,0x00);
    f_time=ti*256+read(0x2E,0x01);
    TMOD=0x21;
    TH0=0xee;
    TL0=0x00;
    TH1=TL1=0xfd;
    SCON=0x50;
    REN=1;
    CMOD=0x00;
    ET0=1;
    EA=1;
    P1M0=0x02;
```

```
    P1M1=0x00;
    ADC_CONTR =0x81;
    TR0=1; TR1=1;
}
void inte(void) interrupt 1
{
    TH0=0xee;
    TL0=0x00;
    five_ms=1;
    if(work==Renwu0_ing||work==Renwu1_ing)
    Run_time++;                          //行驶时间计算
    else Run_time=0;
    Find_Way();                          //寻迹
    if(delay!=0)
    { delay--; }
    if(Sound_time!=0)
    {
        Sound=0;
        Sound_time--;
    }
    else Sound=1;
}
```

本 章 小 结

学习单片机的根本目的就是应用单片机解决实际问题，本章主要学习单片机应用系统开发设计的一般步骤与方法，并通过简易电脑时钟、数字电压表和小车自动平衡系统实际例子了解单片机的应用，了解可靠性设计的基本要求。本章要求：了解单片机应用系统的一般性能要求；掌握单片机应用系统的设计步骤，从总体设计到系统成功运行这一全过程；了解单片机应用系统的软硬件抗干扰措施，在设计硬件时如何筛选元件，如何采取隔离、屏蔽措施，如何考虑在印制电路板设计中的问题，在设计程序时如何使用数字滤波技术、指令冗余技术、软件陷井技术等；通过单片机应用实例体会单片机实际应用。

习 题 9

9.1 单片机应用系统设计的一般方法和步骤是什么？

9.2 从元件级设计单片机应用系统时，硬件设计和软件设计主要包含哪些主要内容？

9.3 单片机应用系统的调试步骤是什么？其主要任务又如何？

9.4 干扰主要有哪些来源？

9.5 硬件抗干扰措施主要有哪些？分别对付什么干扰信号？

9.6 软件抗干扰措施主要有哪些？分别对付什么干扰信号？

9.7 查阅相关网站和杂志，综述利用单片机开发智能仪器的基本情况。

附录A　ASCII码字符表

高3位 / 低4位		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DEL	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	“	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	‘	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	—	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	US	/	?	O	←	o	DEL

表中符号说明：

NUL	空	DEL	数据链换码
SOH	标题开始	DC1	设备控制 1
STX	正文结束	DC2	设备控制 2
ETX	本文结束	DC3	设备控制 3
EOT	传输结束	DC4	设备控制 4
ENQ	询问	NAK	否定
ACK	承认	SYN	空转同步
BEL	报警符	ETB	信息组传送结束
BS	退一格	CAN	作废
HT	横向列表	EM	纸尽
LF	换行	SUB	减
VT	垂直列表	ESC	换码
FF	走纸控制	FS	文字分隔符
CR	回车	GS	组分隔符
SO	移位输出	RS	记录分隔符
SI	移位输入	US	单元分隔符
SP	空格	DEL	作废

附录B　MCS-51 单片机指令表

续表

指令名称和助记符	机器码	指令功能	P	OV	AC	Cy	字节	周期
ACALL addr11	01H	(PC)←(PC)+1 (SP)←(SP)+1 ((SP))←$(PC)_{7\sim0}$ (SP)←(SP)+1 ((SP))←$(PC)_{15\sim8}$ $(PC)_{10\sim0}$←$addr_{10\sim8}$	×	×	×	×	2	2
ADD A，Rn	28H～2FH	(A)←(A)+(Rn)	√	√	√	√	1	1
ADD A，direct	25H	(A)←(A)+(direct)	√	√	√	√	2	1
ADD A，@Ri	26H～27H	(A)←(A)+((Ri))	√	√	√	√	1	1
ADD A，#data	24H	(A)←(A)+data	√	√	√	√	2	1
ADD A，Rn	38H～3FH	(A)←(A)+(Rn)+Cy	√	√	√	√	1	1
ADDC A，direct	35H	(A)←(A)+(direct)+Cy	√	√	√	√	2	1
ADDC A，@Ri	36H～37H	(A)←(A)+((Ri))+Cy	√	√	√	√	1	1
ADDC A，#data	34H	(A)←(A)+data←Cy	√	√	√	√	2	1
AJMP addr11		(PC)←(PC)+2 $(PC)_{10\sim0}$←addr11	×	×	×	×	2	2
ANL A，Rn	58H～5FH	(A)←(A)∧(Rn)	√	×	×	×	1	1
ANL A，direct	55H	(A)←(A)∧(direct)	√	×	×	×	2	1
ANL A，@Ri	56H～57H	(A)←(A)∧((Ri))		×	×	×	1	1
ANL A，#data	54H	(A)←(A)∧data	√	×	×	×	2	1
ANL direct，A	52H	(direct)←(A)∧(direct)	×	×	×	×	2	1
ANL direct，#data	53H	(direct)←(direct)∧data	×	×	×	×	3	2
ANL C，bit	82H	Cy∧(bit)	×	×	×	√	2	2
ANL C，/bit	B0H	Cy←Cy∧($\overline{bit}$)	×	×	×	√	2	2
CJNE A，direct，rel	B5H	若(A)=(direct)，则 (PC)←(PC)+3，Cy←0 若 (A)>(direct)，则 (PC)←(PC)+3+rel， Cy←0 若(A)<(direct)，则 (PC)←(PC)+3+rel Cy←1	×	×	×	×	3	2
CJNE A，#data，rel	B4H	若 (A)=data，则 (PC)←(PC)+3，Cy←0 若 (A)>data，则 (PC)←(PC)+3+rel，	×	×	×	×	3	2

续表

指令名称和助记符	机器码	指令功能	P	OV	AC	Cy	字节	周期
		Cy←0 若 (A)<data，则 (PC)←(PC)+3+rel Cy←1						
CJNE Rn，#data，rel	B8H～BFH	若 (Rn)=data，则 (PC)←(PC)+3，Cy←0 若 (Rn)>data，则 (PC)←(PC)+3+rel CY←0 若 (Rn)<data，则 (PC)←(PC)+3+rel CY←1	×	×	×	×	3	2
CJNE @Ri，#data，rel	B6H～B7H	若 ((Ri))=data，则 (PC)←(PC)+3，CY←0 若 ((Ri))>data，则 (PC)←(PC)+3+rel， Cy←0 若 ((Ri))<data，则 (PC)←(PC)+3+rel Cy←1	×	×	×	×	3	2
CLR A	E4H	(A)←0	√	×	×	×	1	1
CLR C	C3H	Cy←0	×	×	×	√	1	1
CLR bit	C2H	(bit)←0	×	×	×	×	2	1
CPL A	F4H	(A)←($\overline{A}$)	×	×	×	×	1	1
CPL C	B3H	(C)←($\overline{C}$)	×	×	×	√	1	1
CPL bit	B2H	(bit)←($\overline{bit}$)	×	×	×	×	2	1
DA A	D4H	对(A)进行十进制调整	√	×	√	√	1	1
DEC A	14H	(A)←(A)–1	√	×	×	×	1	1
DEC Rn	18H～1FH	(Rn)←(Rn)–1	×	×	×	×	1	1
DEC direct	15H	(direct)←(direct) –1	×	×	×	×	2	1
DEC @Ri	16H～17H	(Ri)←((Ri)) –1	×	×	×	×	1	1
DIV AB	84H	(A)←(A)/(B)的商 (B)←(A)/(B)的余数	√	√	√	√	1	4
DJNZ Rn，rel	D8H～DFH	(Rn)←(Rn) –1 若 (Rn)≠0，则 (PC)←(PC)+2+rel 若 (Rn)=0，则 (PC)←(PC)+2	×	×	×	×	3	2
DJNZ direct，rel	D5H	(direct)←(direct)-1 若 (direct) ≠0，则 (PC)←(PC)+3+rel 若 (direct)=0，则 (PC)←(PC)+3	×	×	×	×	3	2

续表

指令名称和助记符	机器码	指令功能	P	OV	AC	Cy	字节	周期
INC A	04H	(A)←(A)+1	√	×	×	×	1	1
INC Rn	08H～0FH	(Rn)←(Rn)+1	×	×	×	×	1	1
INC direct	05H	(direct)←(direct)+1	×	×	×	×	2	1
INC @Ri	05H～07H	((Ri))←((Ri))+1	×	×	×	×	1	1
INC DPTR	A3H	(DPTR)←(DPTR)+1	×	×	×	×	1	2
JB bit，rel	20H	若 (bit)=1，则 (PC)←(PC)+3+rel 若 (bit) ≠1，则 (PC)←(PC)+3	×	×	×	×	3	2
JBC bit，rel	10H	若 (bit)=1，则 (PC)←(PC)+2+rel， (bit)←0 若 (bit) ≠1，则 (PC)←(PC)+3	×	×	×	×	3	2
JC rel	40H	若(C)=1，则 (PC)←(PC)+2+rel 若(C) ≠1，则 (PC)←(PC)+2	×	×	×	×	2	2
JMP @A+DPTR	73H	(PC)←(A)+(DPTR)	×	×	×	×	1	2
JNB bit，rel	30H	若 (bit)=0，则 (PC)←(PC)+3+rel 若 (bit) ≠0，则 (PC)←(PC)，则 (PC)←(PC)+3	×	×	×	×	3	2
JNC rel	50H	若 (C)=0，则 (PC)←(PC)+2+rel 若 (C) ≠0，则 (PC)←(PC)+2	×	×	×	×	2	2
JNZ rel	70H	若 (A)=0，则 (PC)←(PC)+2+rel 若 (A) ≠0，则 (PC)←(PC)+2	×	×	×	×	2	2
JZ rel	60H	若 (A)=0，则 (PC)←(PC)+2+rel 若 (A) ≠0，则 (PC)←(PC)+2	×	×	×	×	2	2
LCALL addr16	12H	(PC)←(PC)+3 (SP)←(SP)+1 (SP)←$(PC)_{7\sim0}$ (SP)←(SP)+1 (SP)←$(PC)_{15\sim8}$ (PC)←addr16	×	×	×	×	3	2
LJMP addr16	02H	(PC)←addr16	×	×	×	×	3	2

续表

指令名称和助记符	机器码	指令功能	P	OV	AC	Cy	字节	周期
MOV A，Rn	E8H～EFH	(A)←(Rn)	√	×	×	×	1	1
MOV A，direct	E5H	(A)←(direct)	√	×	×	×	2	1
MOV A，@Ri	E6H～E7H	(A)←((Ri))	√	×	×	×	1	1
MOV A，#data	74H	(A)←data	√	×	×	×	2	1
MOV Rn，A	F8H～FFH	(Rn)←(A)	×	×	×	×	1	1
MOV Rn，direct	A8H～AFH	(Rn)←(direct)	×	×	×	×	2	2
MOV Rn，#data	78H～7FH	(Rn)←(data)	×	×	×	×	2	1
MOV direct，A	F5H	(direct)←(A)	×	×	×	×	2	1
MOV direct，Rn	88H～8FH	(direct)←(Rn)	×	×	×	×	2	2
MOV direct1，direct2	85H	(direct2)←(direct1)	×	×	×	×	3	2
MOV direct，@Ri	86H～87H	(direct)←((Ri))	×	×	×	×	2	2
MOV direct，#data	75H	(direct)←data	×	×	×	×	3	2
MOV @Ri，A	F6H～F7H	((Ri))←(A)	×	×	×	×	1	1
MOV @Ri，direct	A6H～A7H	((Ri))←(direct)	×	×	×	×	2	2
MOV @Ri，#data	76H～77H	((Ri))←data	×	×	×	×	2	1
MOV C，bit	A2H	(C)←(bit)	×	×	×	√	2	1
MOV bit，C	92H	(bit)←(C)	×	×	×	×	2	2
MOV DPTR，#data16	90H	DPH←$data_{15\sim8}$ DPL←$data_{7\sim0}$	×	×	×	×	3	2
MOVC A，@A+DPTR	93H	(A)←((A)+(DPTR))	√	×	×	×	1	2
MOVC A，@A+PC	83H	(A)←((A)+(PC))	√	×	×	×	1	2
MOVX A，@Ri	E2H～E3H	(A)←((Ri))	√	×	×	×	1	2
MOVX A，@DPTR	E0H	(A)←((DPTR))	√	×	×	×	1	2
MOVX @Ri，A	F2H～F3H	((Ri))←(A)	×	×	×	×	1	2
MOVX @DPTR，A	F0H	(DPTR)←(A)	×	×	×	×	1	2
MUL AB	A4H	(B)(A)←(A)×(B)	√	√	×	√	1	4
NOP	00H	(PC)←(PC)+1	×	×	×	×	1	1
ORL A，RN	48H～4FH	(A)←(A)∨(Rn)	√	×	×	×	1	1
ORL A，direct	45H	(A)←(A)∨(direct)	√	×	×	×	2	1
ORL A，@Ri	46H～47H	(A)←(A)∨((Ri))	√	×	×	×	1	1
ORL A，#data	44H	(A)←(A)∨data	√	×	×	×	2	1
ORL direct，A	42H	(direct)←(direct)∨(A)	×	×	×	×	2	1
ORL direct，#data	43H	(direct)←(direct)∨data	×	×	×	×	3	2
ORL C，bit	72H	(C)←(C)∨(bit)	×	×	×	√	2	2
ORL C，/bit	A0H	(C)←(C)∨($\overline{bit}$)	×	×	×	√	2	2
POP direct	D0H	(direct)←((SP)) (SP)←(SP)–1	×	×	×	×	2	2
PUSH direct	C0H	(SP)←(SP)+1 ((SP))←(direct)	×	×	×	×	2	2
RET	22H	$(PC)_{15\sim8}$←((SP)) SP←(SP)–1 $(PC)_{7\sim0}$←((SP))	×	×	×	×	1	2

续表

指令名称和助记符	机器码	指令功能	P	OV	AC	Cy	字节	周期
		SP←(SP)–1						
RETI	32H	$(PC)_{15\sim8}$←((SP)) (SP)←(SP)–1 $(PC)_{7\sim0}$←((SP)) (SP)←(SP) –1	×	×	×	×	1	2
RL A	23H	(A)n+1←(A)n $(A)_0$←$(A)_7$	×	×	×	×	1	1
RLC A	33H	$(A)_0$←$(A)_7$ $(A)_0$←(C) (C)←$(A)_7$	√	×	×	√	1	1
RR A	03H	An←(An+1) A_7←(A_0)	×	×	×	×	1	1
RRC A	13H	(A)n←(A)n+1 $(A)_7$←(C) (C)←$(A)_0$	√	×	×	√	1	1
SETB C	D3H	(C)←1	×	×	×	√	1	1
SETB bit	D2H	(bit)←1	×	×	×	×	2	1
SJMP rel	80H	(PC)←(PC)+2 (PC)←(PC)+rel	×	×	×	×	2	2
SUBB A，Rn	98H～9FH	(A)←(A) – (Rn)-Cy	√	√	√	√	1	1
SUBB A，direct	95H	(A)←(A) – (direct)-Cy	√	√	√	√	2	1
SUBB A，@Ri	96～97H	(A)←(A) – ((Ri))-Cy	√	√	√	√	1	1
SUBB A，#data	94H	(A)←(A) –data-Cy	√	√	√	√	2	1
SWAP A	C4H	$(A)_{7\sim4}$↔ $(A)_{3\sim0}$	×	×	×	×	1	1
XCH A，RN	C8H～CFH	(A) ↔ (Rn)	√	×	×	×	1	1
XCH A，direct	C5H	(A) ↔ (direct)	√	×	×	×	2	1
XCH A，@Ri	C6H～C7H	(A) ↔ ((Ri))	√	×	×	×	1	1
XCHD A，@Ri	D6H～D7H	$(A)_{3\sim0}$↔⊕$((Ri))_{3\sim0}$	√	×	×	×	1	1
XRL A，Rn	68H～6FH	A←(A) ⊕ (Rn)	√	×	×	×	1	1
XRL A，direct	65H	A←(A) ⊕ (direct)	√	×	×	×	2	1
XRL A，@Ri	66H～67H	A←(A) ⊕ ((Ri))	√	×	×	×	1	1
XRL A，#data	64H	A←(A) ⊕data	√	×	×	×	2	1
XRL direct，A	62H	Direct←(direct) ⊕ (A)	×	×	×	×	2	2
XRL direct，#data	63H	Direct←(direct) ⊕data	×	×	×	×	3	2

注：P 表示指令执行对奇偶标志的影响，有影响为“√”，无影响为“×”。

OV 表示指令执行对溢出标志位的影响，有影响为“√”，无影响为“×”。

AC 表示指令执行对辅助进位标志位的影响，有影响为“√”，无影响为“×”。

Cy 表示指令执行对进位标志位的影响，有影响为“√”，无影响为“×”。

参 考 文 献

[1] 何立民．单片机应用系统设计[M]．北京：北京航空航天大学出版社，1995
[2] 徐仁贵，廖哲智．单片微型计算机应用技术[M]．北京：机械工业出版社，2001
[3] 朱宇光主编．单片机应用新技术教程[M]．北京：电子工业出版社，2000
[4] 胡文金．单片机应用技术实训教程[M]．重庆：重庆大学出版社．2005
[5] 张毅刚，彭喜源等. MCS-51 单片机应用设计[M]. 哈尔滨：哈尔滨工业大学出版社，1997
[6] 李朝青．单片机原理及接口技术[M]．北京：北京航空航天大学出版社，1999
[7] 张淑清，姜万录等. 单片微型计算机接口技术及其应用[M]. 北京：国防工业出版社，2001
[8] 张慰兮．微型计算机原理、接口及应用[M]．南京：南京大学出版社，1999
[9] 王福瑞等．单片微机测控系统大全[M]．北京：北京航空航天大学出版社，1999
[10] 房小翠，王金凤. 单片机实用系统设计技术[M]. 北京：北京航空航天大学出版社，1999
[11] 曹琳梨，曹巧媛．单片机原理及接口技术[M]．长沙：国防科技大学出版社，2000
[12] 丁元杰．单片机原理及应用[M]．北京：机械工业出版社，1997
[13] 马忠梅等. 单片机的 C 语言应用程序设计[M]. 北京：北京航空航天大学出版社，1999
[14] 肖洪兵，胡辉，郭速学．跟我学用单片机[M]．北京：北京航空航天大学出版社，2002
[15] 闫玉德，俞虹．MCS-51 单片机原理与应用(C 语言版) [M]．北京：机械工业出版社，2003
[16] 杨忠煌，黄博俊，李文昌．单芯片 8051 实务与应用[M]．北京：中国水利水电出版社，2001
[17] 马淑华，王凤文，张美金．单片机原理与接口技术[M]．北京：北京邮电大学出版社，2005
[18] 黄晓明．基于单片机“看门狗”抗干扰技术的探讨[J]．湖北：湖北教育学院学报，2006.8
[19] 陈龙三．8051 单片机 C 语言控制与应用[M]．北京：清华大学出版社，1999
[20] 周立功等. 增强型 80C51 单片机速成与实践[M]. 北京：北京航空航天大学出版社，2003